Applications of Finite Model Theory: Optimisation Problems, Hybrid Modal Logics and Games.

GATE, JAMES,SIMON

Abstract

There exists an interesting relationships between two seemingly distinct fields: logic from the field of Model Theory, which deals with the truth of statements about discrete structures; and Computational Complexity, which deals with the classification of problems by how much of a particular computer resource is required in order to compute a solution. This relationship is known as Descriptive Complexity and it is the primary application of the tools from Model Theory when they are restricted to the finite; this restriction is commonly called Finite Model Theory.

In this thesis, we investigate the extension of the results of Descriptive Complexity from classes of decision problems to classes of optimisation problems. When dealing with decision problems the natural mapping from true and false in logic to yes and no instances of a problem is used but when dealing with optimisation problems, other features of a logic need to be used. We investigate what these features are and provide results in the form of logical frameworks that can be used for describing optimisation problems in particular classes, building on the existing research into this area.

Another application of Finite Model Theory that this thesis investigates is the relative expressiveness of various fragments of an extension of modal logic called hybrid modal logic. This is achieved through taking the Ehrenfeucht-Fraïssé game from Model Theory and modifying it so that it can be applied to hybrid modal logic. Then, by developing winning strategies for the players in the game, results are obtained that show strict hierarchies of expressiveness for fragments of hybrid modal logic that are generated by varying the quantifier depth and the number of proposition and nominal symbols available.

Applications of Finite Model Theory

Optimisation Problems, Hybrid Modal Logics and Games

James S. Gate

supervised by

Professor Iain A. Stewart and Doctor Stefan Dantchev

Thesis submitted for the degree of Doctor of Philosophy

School of Engineering and Computer Sciences

Durham University, UK

2012.

Contents

List of Figures

Declaration

The work contained within this thesis represents the original research of the author under the supervision of Prof. Iain Stewart, with the exception of the results from Section 3.3, which were reached in collaboration with Dr. Prabhu Manyem and Prof. Iain A. Stewart. Results from Sections 3.3 and 3.4 have been published in [GS10].

No part of this thesis has previously been submitted for any degree at any institution.

Statement of Copyright

Acknowledgements

First and foremost I would like to thank my supervisor, Professor Iain A. Stewart, for igniting my interest in Finite Model Theory and enabling me to study it through this PhD. I would also like to thank the EPSRC for providing the necessary financial support to study and University College, Durham for providing a social environment to study within. I wish also to acknowledge the various other students and staff within the Engineering and Computing Science's Algorithms and Complexity group, for everything from listening to presentations on and discussing my research; through escaping the office over a coffee; to putting the world to rights over a pint or two. They include, in no particular order: Dr. Pim van 't Hof, Dr. Luke Mathieson, Dr. Lars Nagel, Ioannis Lignos, Dr. Mark Rhodes, Dr. Barnaby Martin, Dr. Ross Kang, Dr. Berndt Muller, Dr. Yonghong Xiang, Dr. Anna Huber and I am sure many others.

I also wish to thank everyone I met, talking with and listened to at the various conferences I attended in Bristol, Birmingham, Edinburgh, Warwick, Lipari and Kazan. Special thanks should go to Dr. Florent Madelaine and Professor Arnaud Durand for helping me organise research talks at both Clermont-Ferrand and Paris VII and for the hospitality they provided.

Last but by no means least, I would like to thank my partner, Dr. Amy Jordan, for her help, support and love.

Dedication

Dedicated to my parents Janet and Martin and to Amy.

Chapter 1

Introduction

Within the last century the computer has developed from little more than a mathematical concept and general curiosity to an incredibly powerful and useful tool that appears in almost every walk of life all over the world; in this time it has moved from being no more powerful than an abacus to being able to perform quadrillions of computations every second [Top12]. Whilst these huge increases in speed have allowed computers to solve increasingly larger and more complex problems, the laws that underpin their limitations haven't changed. For example, most supercomputers rely on combining multiple processors in parallel and so, if presented with a problem that is inherently sequential they will not be able to efficiently solve it. Another example is a problem for which the most efficient algorithm runs in a number of steps that is exponential as a function of the size of the input; while it may be computable in a reasonable amount of time for a small problem size, any increase in the problem size quickly renders the problem computable in excessively large amounts of time due to the exponential growth of the function. This is where the field of Theoretical Computer Science comes in.

Theoretical Computer Science studies the underlying principles of computation and aims to provide tools for the analysis and classification of computational problems; as opposed to constructing working computers (Electrical Engineering) or developing tools for using them (Software Engineering). Its primary focus over the years has been to classify problems based on the computational resources (mainly processor time and memory) they require and then study the relationships between these classes.

In order to build up a machine independent view of computation, Theoretical Computer Science distances itself from the physical machine by, among other methods, considering problems as simply questions. Such questions are posed in natural language, for example: "What is the shortest distance from Durham to Glasgow?" is a problem whose answer is the shortest distance between the two cities; it has as its input a map and a pair of cities (in this case Durham and Glasgow) and then the goal is to find the shortest distance between them on the map.

A computational method for solving a problem is called an *algorithm*, which is a sequence of computational steps that transforms the input to the problem into an answer to the question. Now, since there are multiple algorithms for solving the same problem, it is interesting to analyse how much of a computer's resources they use in order to compare them. The two primary computer resources that are measured in this analysis are *time*

(the number of computational steps needed) and *space* (the amount of memory required). Other measures, such as the number of processors used when studying algorithms running on parallel systems, are of interest but are outside the scope of this thesis and so are not considered here. By examining the relationship between the resources used and the size of an instance of a problem the relative complexity of an algorithm can be found and compared to other algorithms for the same problem and also for different problems.

The comparison of the resources used by different algorithms is not the main application of Theoretical Computer Science; it is a tool used to study the relative complexity of different problems. The fundamental concept here is that there exists a "best" algorithm for solving a certain problem, in that there is no "better" algorithm that uses significantly less computational resources. By giving an algorithm that solves a problem, this algorithm has, in effect, shown an *upper bound* on the complexity of a problem, but what is really interesting is what the *lower bound* for the complexity of the problem is, that is, the point where no computationally more efficient algorithms exist (in fact, this is the point where the upper bound equals the lower bound). The difficulty comes in showing that an algorithm is in fact the best possible one for solving a problem as it requires the consideration of every possible algorithm. To do this in a rigorous mathematical way requires the formalisation of the model of computation and the algorithms that it can run; one such formalisation is the ubiquitous Turing Machine developed by Alan Turing (see, e.g. [Sip06]).

Whilst progress has been made towards classifying problems into various classes and defining which problems are the hardest in those classes through, for example, the theory of NP-completeness [GJ79], very little is known about the relationships between the complexity classes. For example, the infamous problem in Theoretical Computer Science is the question as to whether $\mathsf{P} \subset \mathsf{NP}$, or $\mathsf{P} = \mathsf{NP}$ (see, e.g. [Pap94]).

To be able to recognise and cope with problems that are fundamentally too hard for computers to be able to solve in reasonable amounts of time or space, it is very important to develop these tools through study of and research in the field of Theoretical Computer Science.

1.1 Applications of Finite Model Theory

As mentioned above the study of the complexity of algorithms and problems requires a mathematical formalisation of computation. This is often achieved using the model of computation called the Turing Machine. Whilst this model is very intuitive when thinking about actual physical computers, it falls short when analysing the complexity of problems, as the machine itself has no built-in concept of the complexity of problems and classes thereof. Finite Model Theory on the other hand does.

Finite Model Theory provides a restricted model of computation in which only problems of a particular complexity can be defined using a particular language (logic). This has made the link between a problem and its computational complexity much more solid, as a problem, and not just an algorithm, has a particular complexity if it can be defined using a formula of some logic; this formalism equates *computational complexity classes* to *classes of logical formulae*. Since Finite Model Theory has grown out of mathematical

Model Theory it retains (some of) the tools from this field, such as Ehrenfeucht-Fraïssé games, which has given it powerful methods for analysing the complexity of problems; comparing two complexity classes is the same as comparing the relative expressiveness of formulae from two logics.

Since its conception, Finite Model Theory has been applied to Theoretical Computer Science in many different ways. Here, two applications are presented:

1. Logical frameworks for optimisation problems (Chapter 3).

2. Relative expressiveness of fragments of hybrid modal logic (Chapter 4).

Whilst the areas are not directly linked, as in one does not build upon the other, they both use tools from Finite Model Theory, which was the aim of the research carried out for this thesis. The first application uses features of logics to define optimisation problems and the second uses Ehrenfeucht-Fraïssé style games to prove the existence of strict hierarchies of expressiveness within hybrid modal logic.

1.2 Thesis Layout

This thesis starts by presenting the required preliminaries in Chapter 2. These are broken down into the following key areas:

- Section 2.1 covers general definitions used by both applications.
- Section 2.2 covers the definitions relevant to the research into the logical frameworks for optimisation problems presented in Chapter 3.
- Section 2.3 covers the definitions relevant to the research into the expressiveness of fragments of hybrid modal logic in Chapter 4.

Following on from the preliminaries are Chapters 3 and 4 that each examine different applications of finite model theory. Finally, the thesis closes with Chapter 5, which discusses the results obtained, looks at the open questions they raise and hypothesises about future avenues of research.

1.3 Original Contribution

Of the results presented within this thesis, the following theorems and their corollaries comprise the significant original contribution made by this research to the field of Theoretical Computer Science:

- NP-hardness proof of the minimisation and maximisation frameworks suggested by Manyem and Bueno in [Man08, BM08] (Theorems 3.3.4 and 3.3.6).
- Frameworks for characterising polynomially-bounded P-optimisation problems;
  - Using fixed-point logic (Theorem 3.4.1);
  - Using second-order Horn logic (Theorem 3.6.6).

- Ehrenfeucht-Fraïssé style games for hybrid modal logic with one modality and universal access (Theorems 4.4.4 and 4.4.7).
- Hierarchies of expressiveness of fragments of Hybrid Graph Logic, which is a hybrid modal logic with one modality, as presented in [BS09] (Theorems 4.5.2, 4.5.6, 4.5.9 and 4.5.10).

A more thorough discussion of the original contribution of these results is made in Chapter 5.

Chapter 2

Preliminaries

In this chapter, the common definitions that will be used throughout this thesis are laid out in Section 2.1, along with those that are specific to the two main chapters: Section 2.2 contains definitions for Chapter 3 and Section 2.3 for Chapter 4. Whilst these preliminaries are intended to be self contained, at the beginning of each section the reader is pointed towards the key textbooks that contain a more in-depth discussion of the area.

2.1 General Definitions and Theorems

The general theme of this thesis is the application of finite model theory to computational complexity and algorithms, a field broadly known as descriptive complexity. Finite model theory is introduced in Section 2.1.1 followed by computational complexity theory in Section 2.1.2, before outlining the major results of descriptive complexity in Section 2.1.4. In the interim, one important class of problems, satisfiability problems, is introduced in Section 2.1.3. Methods and tools for showing inexpressibility of a problem in a logic are presented in Section 2.1.5.

A good introduction to classical mathematical logic is [HA50]; the seminal text on the restrictions of mathematical model theory to the finite is [EF99], although equally good and sometimes regarded as clearer introductions can be found in [Lib04, Kol07, Grä07]. Note that most books on finite model theory focus quite heavily on its applications to computational complexity; no more so than the aptly titled *Descriptive Complexity* [Imm99].

The theory of computation and its complexity is presented in many textbooks such as [Sip06], although notably the relationships between this and mathematical logic through finite model theory are presented from the computational perspective in [Pap94, BBJ02]. A good general textbook on algorithms is [CLRS09], with those particular to logic analysed more closely in [BL99] and the theory of NP-completeness is presented in Garey and Johnson's seminal work [GJ79].

2.1.1 Finite Model Theory

Mathematical model theory is concerned with models, which are in essence discrete data structures, and asks whether a particular property of a model, or a class of models, can be written using a sentence in some logic. In order for a logic to talk about a structure, it

must be the case that they are both of the same *vocabulary*. A vocabulary τ, consists of the relations, functions and constants that comprise a discrete structure and that a formula in some logic can use to talk about and describe the properties of these structures.

Definition 2.1.1. A vocabulary τ is a sequence of r relation symbols, s function symbols and t constant symbols. In general:

$$\tau = \langle R_1^{a_1}, ..., R_r^{a_r}, f_1^{b_1}, ..., f_s^{b_s}, c_1, ..., c_t \rangle$$

where each relation symbol $R_i^{a_i}$ is of arity a_i and each function symbol $f_j^{b_j}$ is of arity b_j. Note that whilst constant symbols are sometimes represented as functions of arity zero, here they are treated as separate symbols with no concept of arity.

With the vocabulary fixed, it is now possible to talk about a structure that is of a particular vocabulary. Every structure has a universe of available objects and an assignment from each symbol in its vocabulary (be it a relation, function or constant) to an appropriate construct of the objects in the universe.

Definition 2.1.2. A *τ-structure* (or sometimes a *τ-model*) $\mathcal{A}$ of vocabulary τ (see Definition 2.1.1) is in general defined as:

$$\mathcal{A} = \langle A, R_1^{\mathcal{A}}, ..., R_r^{\mathcal{A}}, f_1^{\mathcal{A}}, ..., f_s^{\mathcal{A}}, c_1^{\mathcal{A}}, ..., c_t^{\mathcal{A}} \rangle$$

where A is a unary set of objects called the *universe of* $\mathcal{A}$ and each symbol from the vocabulary is *instantiated* as follows:

i) each relation symbol $R_i^{a_i}$ is instantiated as $R_i^{\mathcal{A}} \subseteq A^{a_i}$ (where a_i is the arity of the i^{th} relation symbol)

ii) each function symbol $f_j^{b_j}$ is instantiated as a total function $f_j^{\mathcal{A}} : A^{b_j} \to A$ (again, where b_j is the arity of the j^{th} function symbol)

iii) each constant symbol c_k is instantiated as $c_k^{\mathcal{A}} \in A$.

Note that the arities of the instances of relation and function symbols ($R_i^{\mathcal{A}}$ and $f_j^{\mathcal{A}}$ respectively) are omitted from the notation. This is because the arity can be derived from the size of the tuples in the particular instance: if $R_i^{\mathcal{A}}$ contains only l-tuples then R_i is of arity l; if $f_j^{\mathcal{A}}$ contains only mappings from $\bar{a} \mapsto b$, where $\bar{a}$ is an m-tuple, then f_j is an m-ary function.

Definition 2.1.3. The class of all τ-structures, that is all structures of a particular vocabulary τ, is denoted as STRUCT[τ].

Remark 2.1.4. When the universe of a structure is a finite set the structure is also said to be finite. From now onwards in this thesis all structures, unless otherwise stated, are finite.

Now that the formal definitions of a vocabulary and a structure have been stated, it serves to demonstrate their usage through an example, in order to explain the short-hands and abbreviations that are regularly used.

Example 2.1.5 (The Vocabulary of Graphs)**.** A *directed graph*, often called a *digraph,* (see [Die97, Bol98, AW00]) is a common structure used in computation and especially in the theory thereof. It consists of a set of *vertices* and a set of *edges* that link the vertices; it is commonly represented as a domain V of vertices and a binary set E of edges in the structure $G = \langle V, E \rangle$.

An *undirected graph*, or simply a *graph*, is a restriction of the more general *digraph* in which the existence of an edge between two vertices u and v implies there is also an edge between v and u.

In order to represent a graph as a structure, fix the vocabulary to $\tau_G = \langle E^2 \rangle$ that is, the vocabulary consists of just one binary relation symbol E. An instance of a structure in this vocabulary is $\mathcal{H} = \langle H, E^{\mathcal{H}} \rangle$, where $H = V$ and $E^{\mathcal{H}} = E$ from the graph structure G in the previous paragraph. It should be clear to see that $\mathcal{H}$ and G represent the same graph and that there is a one-to-one mapping between any graph structure and its representation as a τ_G-structure.

Now onto the short-hand: when the context that a symbol is presented in makes it implicit as to whether it is a vocabulary symbol (such as E^2) or an instance of one (such as $E^{\mathcal{H}}$) that is being talked about then the superscript is omitted. As an example consider the graph edge relation E. When E is mentioned on its own it is the edge relation in general, and hence the vocabulary symbol E^2 that is being talked out. Whereas when a question such as, "is there an edge between vertices a and b?" is asked, it is a particular instance of the edge relation E, say $E^{\mathcal{G}}$, that is being referred to. Since these questions are often asked of all graphs, the superscript is omitted, except when it is required for clarity.

Another notational shorthand used throughout this thesis is that the universe of a structure is written using a Roman capital letter (e.g. H) whereas the actual structure itself is written using a calligraphic capital letter (e.g. $\mathcal{H}$).

Example 2.1.6. Another common vocabulary is $\tau_{Gst} = \langle E^2, s, t \rangle$. It is used to represent a graph plus two named vertices, the *start vertex* s and the *end vertex* t.

Now that the concept of a vocabulary and of the structures that use it have been defined, it remains to define a language that can be used to describe properties of these structures. The first such language that shall be introduced is *first-order logic*. Although, as will later be explained, it is not a language expressive enough to be of specific interest, it is the basis of almost all other languages that are and as such it is a useful starting point for demonstrating how a language can be used to describe properties of structures.

From now onwards only *relational vocabularies* shall be considered, that is vocabularies without any function symbols. Under such a vocabulary, there are only two types of *atomic formulae*. In a logic, atomic formulae are those whose truth relies solely on the structure and not on other fragments of the formula, since they cannot be decomposed into simpler atoms. A *well-formed formula* is one that is valid according to the definition of the syntax below:

Definition 2.1.7. The syntax of first-order logic in the vocabulary τ is recursively defined as follows:

$$\varphi := R_i(x_1, ..., x_{a_i}) \mid x = y \mid \bot \mid \psi_1 \Rightarrow \psi_2 \mid \exists z \, \psi$$

where $R_i^{a_i} \in \tau$ (that is R_i is a relation symbol of arity a_i) and in the tuple $(x_1, ..., x_{a_i})$, each x_j is either a first-order variable or a constant symbol from τ; $\bot$ is a special symbol that always represents the truth value of FALSE; x and y are each either a first-order variable or a constant symbol from τ; ψ_1 and ψ_2 are both well-formed first-order logic formulae; z is a first-order variable; and ψ is a well-formed first-order formula.

Formulae of the form $R_i(x_1, ..., x_{a_i})$, $x = y$ and $\bot$ are atomic. The $\Rightarrow$ symbol is a boolean operator from first-order *propositional* logic and the $\exists$ symbol is a quantifier from first-order *predicate* logic [HA50].

Definition 2.1.8. The class of all first-order formulae is FO; the class of all first-order formulae of the same vocabulary τ is denoted as FO[τ].

The *free variables* of some formula φ are those that appear in an atomic formula, are not constant symbols from τ and haven't been *bound* by a quantifier. To see this, take the formula $\psi := E(x, y)$: in ψ both x and y are free, but in the formula $\varphi := \exists x\, E(x, y)$ only y is free since the quantifier has bound x. A formula without any free variables is called a *sentence.*

Sometimes it shall be necessary to explicitly state what the free variables of a formula are (often simply for clarity) using a function style notation. For example the formula ψ in the previous example can be written as $\psi(x, y)$, explicitly stating that its free variables are x and y; similarly $\varphi(y)$ explicitly states that y is a free variable in φ.

Free variables can be substituted for concrete values from the universe as follows: $\psi[\frac{a}{x}, \frac{b}{y}]$ substitutes the concrete values a and b for the free variables x and y in the formula ψ, reducing it to the relational query $E(a, b)$. When the free variables of a formula are explicitly stated, as above, then the substitution is implicit: $\psi[\frac{a}{x}, \frac{b}{y}]$ can be written as $\psi(x, y)[a, b]$.

Often it is useful to substitute constant values for free variables. The notation $\psi\,[a/x]$ means "substitute every free occurrence of x with a in the formula ψ" and in the context of the previous example, this gives $\psi\,[a/x] \equiv E(a, y)$. It is often the case that free variables are substituted for constants and so the notation $\psi(a, b)$, where both a and b are constant symbols in the vocabulary τ, is used on a formula $\psi(x, y)$, where both x and y are free variables, as shorthand for $\psi\,[a/x, b/y]$. Note that the statement $\psi(a, b)$ has bound all the free variables in ψ and so is a sentence.

With the syntax defined, the semantic meaning of first-order formulae on a particular structure can be defined.

Definition 2.1.9. Given a first-order formula φ and a structure $\mathcal{A}$ that are both of the same vocabulary τ, it is the case that $\mathcal{A}$ *satisfies* φ (and similarly that $\mathcal{A}$ *is a model of* φ), written $\mathcal{A} \models \varphi$, depending on the form of φ according to the following rules:

- $\mathcal{A} \models R_i(x_1, ..., x_{a_i})$ iff $(x_1, ..., x_{a_i}) \in R_i^{\mathcal{A}}$.
- $\mathcal{A} \models x = y$ iff x and y denote the same member of the universe.
- $\mathcal{A} \models \bot$ never holds, that is $\mathcal{A} \not\models \bot$ for all $\mathcal{A}$.
- $\mathcal{A} \models \psi_1 \Rightarrow \psi_2$ iff when $\mathcal{A} \models \psi_1$ it is also the case that $\mathcal{A} \models \psi_2$.
- $\mathcal{A} \models \exists z\, \psi$ iff there exists some $a \in A$ such that $\mathcal{A} \models \psi\,[a/z]$.

with all the symbols and variables having the same properties as defined in the syntax (Definition 2.1.7).

Definition 2.1.10. The normal boolean operators from propositional logic are constructed using the boolean operator $\Rightarrow$ and the FALSE symbol $\bot$ as follows:

- $\neg\varphi \equiv \varphi \Rightarrow \bot$ (negation)
- $\varphi_1 \vee \varphi_2 \equiv \neg\varphi_1 \Rightarrow \varphi_2$ (or)
- $\varphi_1 \wedge \varphi_2 \equiv \neg(\neg\varphi_1 \vee \neg\varphi_2)$ (and)
- $\varphi_1 \Leftrightarrow \varphi_2 \equiv (\varphi_1 \Rightarrow \varphi_2) \wedge (\varphi_2 \Rightarrow \varphi_1)$ (bi-implication)

The dual of the *existential* quantifier $\exists$, which is the *universal* quantifier $\forall$ is defined as:

- $\forall x\, \varphi \equiv \neg\exists x\, \neg\varphi$

2.1.1.1 Structure Composition

Another feature that is useful to have access to is the ability to compose a structure from a collection of smaller structures. For example, if there is a structure $\mathcal{A}$ of vocabulary τ_1 and a structure $\mathcal{B}$ of vocabulary τ_2, such that $A = B$ (i.e. they are over the same universe) and $\tau_1 \cap \tau_2 = \emptyset$ (i.e. the two vocabularies are disjoint), then the notation $(\mathcal{A}, \mathcal{B}) \models \varphi$ is used to combine the two structures and query whether they satisfy the sentence φ, where φ is appropriate to the vocabulary $\tau_1 \cup \tau_2$. Note that when the two conditions are not met (i.e. $A \neq B$ or $\tau_1 \cap \tau_2 \neq \emptyset$) the notation $(\mathcal{A}, \mathcal{B}) \models \varphi$ is not well-formed, but these conditions can usually be met through a transformation of one of the structures.

Often it is the case that the vocabulary of some of the structures to be composed contains just constant symbols. For example consider a graph structure $\mathcal{G}$ of vocabulary τ_g and a formula φ of vocabulary τ_{gst}. Now the statement $\mathcal{G} \models \varphi$ is not well-formed since s and t do not exist in the structure $\mathcal{G}$, whereas if it is composed with two structures $\mathcal{A}$ and $\mathcal{B}$ of vocabularies $\langle s \rangle$ and $\langle t \rangle$ respectively, then the statement $(\mathcal{G}, \mathcal{A}, \mathcal{B}) \models \varphi$ is well-formed. The notation $(\mathcal{G}, s, t) \models \varphi$ is used as a shorthand when a structure is being composed with structures that contain only constants, as is the case in this example.

2.1.1.2 Logical Equivalence

Although two first-order formulae may differ syntactically, they may agree on all structures. When this is the case these formulae are said to be *logically equivalent*.

Definition 2.1.11. Fix the vocabulary τ. Two FO[τ] formulae, φ and ψ, are *logically equivalent*, denoted $\varphi \equiv_{FO[\tau]} \psi$ iff for every pair of τ-structures $\mathcal{A} \in$ STRUCT[τ] and $\mathcal{B} \in$ STRUCT[τ], it is the case that $\mathcal{A} \models \varphi$ iff $\mathcal{B} \models \psi$.

Every first-order formula is logically equivalent to another formula in which all the quantifiers are in one block at the beginning of the formula and then the rest of it is *quantifier-free*, which is to say that no $\exists$ or $\forall$ symbols appear in it. This particular syntactic form for first-order formula is know as *prenex normal form*.

Definition 2.1.12. A formula φ is in *prenex normal form* if it is in the syntactic form $\varphi := Q_1x_1...Q_rx_r\psi$, where each Q_i is a quantifier ($Q_i \in \{\exists, \forall\}$), each x_i a first-order variable and ψ is quantifier-free. Every first-order formula is logically equivalent to some other first-order formula that is in prenex normal form.

Another type of logical equivalence is between two structures, as opposed to two formulas. Two structures are equivalent in first-order logic when they agree on every sentence.

Definition 2.1.13. Fix the vocabulary τ. Two τ-structures $\mathcal{A}$ and $\mathcal{B}$ are equivalent in first-order logic (or just FO-equivalent), denoted $\mathcal{A} \equiv_{FO} \mathcal{B}$ if, and only if, for every formula $\varphi \in \text{FO}[\tau]$ it is the case that $\mathcal{A} \models \varphi$ iff $\mathcal{B} \models \varphi$.

The concepts of equivalence of formulae and structures in first-order logic can be extended to arbitrary logics; the equivalence relation for a logic $\mathcal{L}$ is denoted $\equiv_{\mathcal{L}}$. Note also that the vocabulary is omitted in Definition 2.1.13 (i.e. FO rather than FO[τ]) since it is implicit from the structures. When talking about the logical equivalence of formulae, the vocabulary is also omitted when it is implicit.

2.1.1.3 Syntactic properties of first-order formulae

Whereas the semantic properties of a formula are dependent on the structure that the formula is being evaluated on, *syntactic properties* (or *metrics*) are those of a formula that can be measured without considering any structures. Most metrics are based on the quantifiers in the formula; one such metric is the *quantifier rank* (sometimes called the *quantifier depth*) of a formula, denoted $\text{qr}(\varphi)$, which is the greatest number of nested quantifications in the formula.

Definition 2.1.14. Given a formula $\varphi \in \text{FO}$, the *quantifier rank* $\text{qr}(\varphi)$ is recursively defined as follows:

(i) If φ is an *atomic formula* (i.e. a relational query, equivalence test or $\bot$) then $\text{qr}(\varphi) = 0$.

(ii) When $\varphi := \psi_1 \Rightarrow \psi_2$, $\text{qr}(\varphi) = \max\{\text{qr}(\psi_1), \text{qr}(\psi_2)\}$.

(iii) When $\varphi := \exists x.\psi$, $\text{qr}(\varphi) = \text{qr}(\psi) + 1$.

where max{...} evaluates to the largest member of a set of integers.

The definitions of quantifier rank for the other boolean operators (see Definition 2.1.10) are derived as follows:

(iv) $\text{qr}(\neg\varphi) = \text{qr}(\varphi \Rightarrow \bot) = \max\{\text{qr}(\varphi), \text{qr}(\bot)\} = \max\{\text{qr}(\varphi), 0\} = \text{qr}(\varphi)$.

(v) $\text{qr}(\psi_1 \vee \psi_2) = \text{qr}(\neg\psi_1 \Rightarrow \psi_2) = \max\{\text{qr}(\neg\psi_1), \text{qr}(\psi_2)\} = \max\{\text{qr}(\psi_1), \text{qr}(\psi_2)\}$.

with the definitions of the two binary boolean operators, $\wedge$ and $\Leftrightarrow$, being of exactly the same format as for $\vee$, i.e. the maximum quantifier rank of the two formulae either side of the operator.

Finally, the universal quantifier is derived as:

(vi) $\text{qr}(\forall x\, \psi) = \text{qr}(\neg\exists x\, \neg\psi) = \text{qr}(\exists x\, \neg\psi) = \text{qr}(\exists x\, \psi) = \text{qr}(\psi) + 1$.

The number of variables (either free or bound) used in a formula is another syntactic measure. For example the sentence $\forall x \forall y \, [E(x, y) \Rightarrow E(y, x)]$ contains two bound variables x and y. This and the measure of quantifier rank are used to generate some classes of first-order formulae.

Definition 2.1.15. The following classes are syntactic restrictions of the class FO:

(i) FO^s contains all first-order formulae with at most s variables (where these variables can be either free or bound). Note that $\mathrm{FO}^s \subseteq \mathrm{FO}^{s+1}$.

(ii) FO_r contains all first-order formulae with quantifier rank of at most r; $\varphi \in \mathrm{FO}_r$ iff $\mathrm{qr}(\varphi) \leq r$. Note that $\mathrm{FO}_r \subseteq \mathrm{FO}_{r+1}$.

(iii) $\mathrm{FO}^s_r = \mathrm{FO}^s \cap \mathrm{FO}_r$.

Further classes are defined for first-order formulae in prenex normal form based on the nature of the quantifiers at the front of a formula. The measure here is that of *quantifier alternation*, which is the number of times the quantifiers switch between being existential and universal. Recall from Definition 2.1.12 that first-order formulae in prenex normal form are those of the form:

$$Q_1 x_1 ... Q_r x_r \, \psi \text{ where each } Q_i \in \{\exists, \forall\} \text{and } \psi \text{is quantifier free.}$$

Each maximal range of quantifiers of the same type is called a *quantifier block*; the range of quantifiers $Q_i x_i ... Q_j x_j$ is a maximal quantifier block if, and only if, all quantifiers in the block are of the same type, either $i = 1$ or $Q_{i-1} \neq Q_i$, and either $j = r$ or $Q_j \neq Q_{j+1}$. A formula in prenex normal form is now classified by the number of different maximal quantifier blocks, which is referred to as its amount of *quantifier alternation*.

Definition 2.1.16. Given a first-order formula φ that is in prenex normal form, let m be the number of maximal quantifier blocks in the quantifiers at the beginning of the formula. The following class of first-order formulae are defined:

i) Σ_m contains all first-order formulae with m maximal quantifier blocks where the leftmost block is existential.

ii) Π_m contains all first-order formulae with m maximal quantifier blocks where the leftmost block is universal.

iii) Δ_m contains all first-order formulae that are logically equivalent to both a formula in Σ_m and a formula in Π_m.

A formula from the class Σ_m is called a Σ_m-formula (similarly for Π_m and Δ_m). Note that when $m = 0$ the classes contain only quantifier free formulae and hence $\Sigma_0 = \Pi_0 = \Delta_0$. Each of the classes forms a hierarchy, that is for $m \geq 0$, $\Sigma_m \subseteq \Sigma_{m+1}$ (similarly for Π and Δ), since a formula with m quantifier alternations can simply add a quantifier with a dummy variable as the rightmost quantifier to generate $m + 1$ alternations.

2.1.1.4 Second-order logic

An extension of first-order logic, called second-order logic, shall now be introduced. It allows quantifiers to operate on variables that are relations over the universe of the structure rather than just single points in the universe.

Definition 2.1.17. The language (syntax) of *second-order logic* is the same as that of first order logic (see Definition 2.1.7) but with the addition of *second-order quantifiers.* These are quantifiers that bind a *relational variable* rather than a first-order variable. Let RVAR $= \langle S_1^{c_1}, ..., S_u^{c_u} \rangle$ be the sequence of u relational (second-order) variables that a formula may use, where each S_i is of arity c_i. It must be the case that RVAR is disjoint from both the symbols in the vocabulary of the formula and the first-order variables (the implicit sequence VAR) that the formula contains.

If φ is a second-order formula then so is $\exists S_i \, \varphi$ and all first-order order formulae are also second-order formulae (but not vice versa). The shorthand $\forall S_i \, \varphi \equiv \neg \exists S_i \, \neg \varphi$ is freely used in second-order formulae. The class of all second-order logic formulae is called SO.

Now with the syntax of second-order logic defined, the semantics can be discussed. The semantics are the same as for first-order logic except when evaluating the second-order quantifier fragment $\exists S_i \, \varphi$. For this the concept of a structure *realising* a sequence of relational variables is required.

Definition 2.1.18. Given a sequence of u relational variables RVAR $= \langle S_1^{c_1}, ..., S_u^{c_u} \rangle$, let the vocabulary $\sigma_{\text{RVAR}} = \text{RVAR}$. A structure $\mathcal{B}$ *realises* the relational variables in RVAR with respect to another structure $\mathcal{A}$ iff it is of the vocabulary σ_{RVAR} and the universe of $\mathcal{B}$ equals the universe of $\mathcal{A}$ (i.e. $B = A$). When RVAR contains only one relation variable, say S, it is said that $\mathcal{B}$ *realises* S.

The semantics of second-order logic use realisation of relational variables to assign them specific values, allowing the resulting first-order formulae to be evaluated as per the semantics of first-order logic (see Definition 2.1.9).

Definition 2.1.19. The semantics of second-order logic are the same as those of first-order logic (see Definition 2.1.9) with the addition of rules for evaluating the second-order quantifiers. Some structure $\mathcal{A}$ satisfies the fragment $\exists S_i \, \varphi$ iff there exists some structure $\mathcal{B}$ that *realises* the relational variable S_i and $(\mathcal{A}, \mathcal{B}) \models \varphi$. It is said that $\mathcal{B}$ *witnesses* S_i in φ. The semantics for the fragment $\forall S_i \, \varphi$ are similarly defined, except that every structure that realises the relational variable S_i must satisfy φ and therefore a single structure cannot witness S_i in φ, instead it can witness that $\forall S_i \, \varphi$ is *unsatisfiable* when $(\mathcal{A}, \mathcal{B}) \not\models \varphi$.

In the same way that first-order logic formulae can be written in prenex normal form (see Definition 2.1.12), every second-order formula is logically equivalent to some second-order formula where all the second-order quantifiers are in a block at the beginning followed by a block of first-order quantifiers.

Definition 2.1.20. Every second-order formula is logically equivalent to a second-order formula of the form:

$$Q_1 X_1 ... Q_m X_m . \psi$$

where each $Q_i \in \{\exists, \forall\}$, each $X_i \in \text{RVAR}$ and ψ is a first-order formula in prenex normal form. The concept of *quantifier blocks* is defined in a manner analogous to that for first-order logic, where it is a maximal sequence of quantifiers such that they are all of the same type.

The number of blocks of the same type of second-order quantifiers gives the class that the formula is in (in a manner similar to the first-order classification in Definition 2.1.16). But for second-order formulae it is parameterised by a second measure: the maximum arity of the relational variables in the second-order quantifier block.

Definition 2.1.21. The class of second-order formulae Σ^r_m contains all second-order formulae in prenex normal form that have relational variables of arity at most r and m blocks of different types of second-order quantifiers, where the first block is an existential one ($\exists$). The class Π^r_m is defined analogously to Σ^r_m, except that the first block is universal ($\forall$). Finally the class Δ^r_m contains all formulae that are logically equivalent to both a Σ^r_m-formula and a Π^r_m-formula.

Definition 2.1.22. The following classes of second-order formulae are used as shorthand for the commonly used classes from Definition 2.1.21:

1. $\text{ESO} = \exists\text{SO} = \bigcup_{r \geq 0} \Sigma^r_1$ *(existential second-order logic)*;
2. $\text{USO} = \forall\text{SO} = \bigcup_{r \geq 0} \Pi^r_1$ *(universal second-order logic)*;
3. $\text{MSO} = \bigcup_{m \geq 0} \Delta^1_m$ *(monadic second-order logic).*

2.1.1.5 Built-in Relations and Orderings

Sometimes, as in the case of descriptive complexity (see Section 2.1.4), a logic that is not expressive enough can be slightly enhanced, by adding to a structure pre-constructed relations and functions that satisfy certain properties before evaluating it against the sentence. Such relations and functions are respectively known as *built-in relations* and *built-in functions*. Here only built-in relations will be discussed since the vocabularies used throughout this thesis are (on the whole) relational.

Definition 2.1.23. A built-in relation is one that is in the vocabulary of a structure/sentence but that is constructed based on the values of other (possibly built-in) relations and constants. There are two important types of built-in relations:

i) *Deterministic:* built-in relations that are always the same for the same structure.

ii) *Non-deterministic:* built-in relations that can vary for the same structure.

In order to introduce the concepts of built-in relations and show how they work, both deterministic and non-deterministic examples are now given:

Example 2.1.24. Let $\sigma_{G^+} = \langle E^2, E^{+^2} \rangle$, that is the vocabulary of digraphs augmented with the binary relation E^+. The relation E^+ is a built-in relation that always contains the transitive closure of E. This means that whenever there are two vertices s and t in

the graph, such that there is a path between s and t over edges defined in E, it is the case that $E^+(s,t)$ holds. Observe that this is a deterministic built-in relation, since there is only one valid E^+ for each E.

More details about the built-in relation defining the transitive closure of the edge relation of a graph, along with its use, can be found in Chapter 4.

Example 2.1.25. Let $\sigma_< = \langle <^2 \rangle$. A τ-structure is said to be *ordered* when it includes the built in ordering relation $<$ from the vocabulary $\sigma_<$, resulting in the ordered $(\tau \cup \sigma_<)$-structure. Using the shorthand $x < y \equiv <(x,y)$, the $<$ relation has the following properties:

(i) $\forall x \forall y \, [x \neq y \Rightarrow (x < y \vee y < x)]$ (the ordering relation is total)

(ii) $\forall x \forall y \, [x < y \Rightarrow \neg \, y < x]$ (the ordering relation is anti-symmetric)

(iii) $\exists! x \forall y \, [x < y]$ (there is exactly 1 element smaller than all other elements)

(iv) $\exists! x \forall y \, [y < x]$ (there is exactly 1 element larger than all other elements)

where $\exists! x \, \varphi$ is the shorthand for $\exists x[\varphi(x) \wedge \forall y(\varphi(y) \Rightarrow x = y))]$ (read: there exists a unique x such that $\varphi(x)$ holds). Observe two facts about the ordering relation $<$: it does not depend on the values of any relations or constants in the original unordered σ-structure and it is non-deterministic. To see the second claim, take an ordered structure $\mathcal{A}$ with universe $A = \{1,2,3\}$. The ordering relations $\{(1,2),(2,3)\}$, $\{(1,3),(3,2)\}$, $\{(2,1),(1,3)\}$, $\{(2,3),(3,1)\}$, $\{(3,1),(1,2)\}$, $\{(3,2),(2,1)\}$ are all valid under the four rules above, meaning that $<$ is a *non-deterministic* built-in relation.

Another type of vocabulary that can be used to make a τ-structure ordered is the *successor relation*, whose vocabulary is given as $\sigma_{succ} = \langle succ^2, min, max \rangle$. The element min is the smallest element in the ordering, max the largest and $succ(x,y)$ means that y comes immediately after x in the ordering. This built-in relation and the two constants have the following properties:

(i) $\forall x[x \neq max \Rightarrow \exists! y \, succ(x,y)]$ (every element except max has exactly one successor)

(ii) $\forall x[x \neq min \Rightarrow \exists! y \, succ(y,x)]$ (every element except min has exactly one predecessor)

(iii) $\neg \exists x \, succ(x, min)$ (min has no predecessor)

(iv) $\neg \exists x \, succ(max, x)$ (max has no successor)

Note that the built-in relation $succ$ and constants min and max are also non-deterministic, and also that the transitive closure of a valid built-in $succ$ relation is a valid built-in $<$ relation.

Definition 2.1.26. A structure is ordered if it has either a *total ordering* of vocabulary $\sigma_<$ or a *successor ordering* of vocabulary σ_{succ} as built-in relations.

The problem with non-deterministic built-in relations, such as ordering, is that they can cause the evaluation of whether a structure satisfies a formula to also be non-deterministic. This is more than often an undesirable property and so the concept of a formula being *invariant* over the different possible values for non-deterministic built-in relations of the same underlying structure is defined.

Definition 2.1.27. A formula φ is *R-invariant*, where R is a non-deterministic built-in relation, when for any $(\sigma \setminus \langle R \rangle)$-structure $\mathcal{A}$ and any two different assignments to R, say R_1 and R_2, $(\mathcal{A}, R_1) \models \varphi \Leftrightarrow (\mathcal{A}, R_2) \models \varphi$.

The most usual type of non-deterministic built-in relation is ordering, so there is a special term for when a formula is invariant over the built-in ordering relations of a structure:

Definition 2.1.28. A formula φ is *order-invariant* when, for any pair of ordered structures $\mathcal{A}$ and $\mathcal{B}$ that both have the same (unordered) universe, it is the case that $\mathcal{A} \models \varphi \Leftrightarrow \mathcal{B} \models \varphi$.

Built-in relations can be used with any logic since they only change the vocabulary of the structure; it is up to the particular logic to define how these built-in relations are accessed, especially if the logic works with a fixed vocabulary or places limitations on the types of vocabulary available.

2.1.1.6 Fixed Points of Operators

Details of fixed point operators, their application to first-order logic and the complexity classes they capture are described in: Chapters 7 and 8 of [EF99]; Chapter 4 (inductive definitions) & Chapter 5 (parallelism) of [Imm99]; Chapter 10 of [Lib04]; and Chapters 2 and 3 of [GKL$^+$07].

The operators of interest here are those that operate on the universe A of some structure $\mathcal{A}$. An *operator on A* is a mapping $F : \mathcal{P}(A) \to \mathcal{P}(A)$, where $\mathcal{P}(A)$ is the powerset of A.

Definition 2.1.29. Given an operator $F : \mathcal{P}(A) \to \mathcal{P}(A)$, a set X (which is an element of the powerset of A) is a fixed-point of F if, and only if, $F(X) = X$.

In the context of finite model theory these operators are used to recursively construct relations. This is achieved by defining a starting point, an iterative step and an end point. For some operator F on A the start point is:

$$X^0 = \emptyset$$

the iterative step is:

$$X^{i+1} = F(X^i)$$

and the end point is the first fixed-point that is encountered.

The operator F is *inductive* when $X^i \subseteq X^{i+1}$ for all i; that is, at each step in the sequence the relation either grows to include more members or stays the same. The value of r when $X^r = X^{r+1}$, i.e. the first fixed point, is the *inductive depth* of the operator F on the structure $\mathcal{A}$.

Definition 2.1.30. Given an operator F on A, a set X (which is an element of the powerset of A) is a *least fixed point* if it is a fixed point and for all Y, where Y is a fixed point of F, it is the case that $X \subseteq Y$.

Furthermore, an operator F on A is *monotone*, if for all X and for all Y, where X and Y are elements of the powerset of A, $X \subseteq Y \Rightarrow F(X) \subseteq F(Y)$, and *inflationary* if $X \subseteq F(X)$.

2.1.1.7 Defining Operators Using Logic

A formula in first-order logic can be used to define an operator. Suppose there is a formula $\varphi(R, \bar{x})$, which is appropriate to the vocabulary τ, with the only free variables being a k-ary relational variable $R \notin \tau$ and a k-tuple $\bar{x}$ of variables. Using this formula, the operator F on A^k is defined as follows:

Definition 2.1.31. Given: a formula $\varphi(R, \bar{x})$ that is appropriate to the vocabulary τ, where R is a k-ary relation variable such that $R \notin \tau$ and $\bar{x}$ is a k-tuple of variables. Then the operator F on A^k that φ gives rise to is defined as:

$$F(R) = \{\bar{u} \mid (\mathcal{A}, \bar{u}) \models \varphi\,(R, \bar{u})\}$$

where $\mathcal{A}$ is a τ-structure whose universe is A and $\bar{u}$ is a k-tuple ranging over A^k.

With this definition, first-order logic can be augmented with fixed-point operators and these can be used to construct relations, starting with the empty set and finishing when a fixed-point is encountered. Before adding this capability to first-order logic, the nature of the fixed-point that is used to indicate the end of this construction needs to be considered.

Theorem 2.1.32 (Tarski-Knaster)**.** *Every monotone operator F has a least fixed point which is equal to the first fixed point encountered in the sequence of relations X^i, which starts with $X^0 = \emptyset$ and has each step inductively defined as $X^{i+1} = F(X^i)$.*

Lemma 2.1.33. *If R only appears positively in $\varphi(R, \bar{x})$ then the operator F that φ gives rise to is monotone.*

A *least fixed point operator* (LFP) is an operator generated from a first-order formula that iteratively constructs a relation, as previously described, stopping at its least fixed point. From the above theorem, it can be seen that if the operator is monotone then it is a least fixed-point operator; moreover if the formula satisfies the hypothesis of Lemma 2.1.33 then the resulting operator is monotone.

The logic FO(LFP) is first-order logic augmented with the capability to define operators, find their least fixed-point and query the resulting relation. The syntax and semantics of FO(LFP) are the same as those of first-order logic, but with the addition of the following rule:

Definition 2.1.34. The logic FO(LFP) has the same syntax and semantics as first-order logic (FO), plus the following syntactic construction: if $\varphi(R, \bar{x})$ is a formula of vocabulary τ, in which $R \notin \tau$ is a k-ary relation variable that only appears positively, then $[\mathrm{LFP}_{R,\bar{x}}\,\varphi](\bar{t})$ is also a formula, where $\bar{x}$ is a k-tuple of variables and $\bar{t}$ is a k-tuple of constants symbols from τ. It is satisfied by a structure $\mathcal{A}$ if the least fixed-point of the operator generated by φ over $\mathcal{A}$ contains the tuple $\bar{t}$.

From now onwards in this thesis the formula $[\mathrm{LFP}_{R,\bar{x}}\,\varphi]$ of the logic FO(LFP) shall be referred to as the *least fixed-point operator* generated by the formula φ. The least fixed-point operator is an operator on A^k, where k is the arity of the relation variable R.

Remark 2.1.35. As first-order logic is defined inductively (see Definition 2.1.7), it follows that with the addition of a new formula to the syntax, the inductive definitions still hold. For example, if ψ is a formula in FO(LFP) then $\psi \wedge [\mathrm{LFP}_{R,\bar{x}}\,\varphi](\bar{t})$ is also a formula. Also, the formula φ that is used to generate the operator is in the logic FO(LFP) and so can itself contain fragments that query the least fixed point of an operator generated from an appropriate formula in FO(LFP).

Strictly, only one least fixed point operator is needed to define a boolean query on a structure. This is due to the following logical equivalence:

Theorem 2.1.36. *Every sentence of the form* $[\mathrm{LFP}_{R,\bar{x}}\,\varphi]\,(\bar{t})$ *where* $\varphi \in \mathrm{FO(LFP)}$*, is logically equivalent to a sentence of the form* $[\mathrm{LFP}_{\mathrm{S},\bar{\mathrm{x}}}\,\psi]\,(\bar{u})$ *where* $\psi \in \mathrm{FO}$*.*

Note that the arities of R and S (and hence the size of $\bar{t}$ and $\bar{u}$) in Theorem 2.1.36 are not necessarily equal. Notwithstanding Theorem 2.1.36, it is still a useful syntactic feature to have multiple nested fixed-point operators, as is demonstrated in the examples in Section 3.5 of Chapter 3.

Another fixed-point operator that will be considered is the *inflationary fixed-point,* or IFP operator. Adding the ability to define and apply this type of operator to first-order logic produces the logic FO(IFP). An IFP operator is defined in a similar way to an LFP operator (see Definition 2.1.34).

Definition 2.1.37. Given a formula $\varphi(R, \bar{x})$ that is appropriate to the vocabulary τ, where R is a k-ary relation variable such that $R \notin \tau$ and $\bar{x}$ is a k-tuple of variables, then the *inflationary fixed point* (IFP) operator F on A^k that φ gives rise to is defined as:

$$F(R) = \{\bar{u} \mid (\mathcal{A}, \bar{u}) \models R(\bar{u}) \vee \varphi(R, \bar{u})\}$$

where $\mathcal{A}$ is a τ-structure whose universe is A and $\bar{u}$ is a k-tuple ranging over A^k.

Corollary 2.1.38. *The operator* F*, as defined in Definition 2.1.37, is inductive.*

In a manner identical to that used to add the least fixed point operator to first-order logic (see Definition 2.1.34), the logic FO(IFP) is defined analogously.

Definition 2.1.39. The syntax and semantics of the logic FO(IFP) are those of first-order (FO) logic, with the addition of the following syntactic construction: if $\varphi(R, \bar{x})$ is a formula of vocabulary τ, in which $R \notin \tau$ is a k-ary relation variable that only appears positively, then $[\mathrm{IFP}_{R,\bar{x}}\,\varphi](\bar{t})$ is also a formula ($\bar{x}$ is a k-tuple of variables and $\bar{t}$ is a k-tuple of constants symbols from τ). Given a structure $\mathcal{A}$, the fixed-point of the IFP operator defined by φ (as per Definition 2.1.37) contains the tuple $\bar{t}$ iff $\mathcal{A} \models [\mathrm{IFP}_{R,\bar{x}}\,\varphi]\,(\bar{t})$.

In the same manner that LFP and IFP operators are added to first-order logic allowing the construction of relations, they can also be added to second-order logic (see Definitions 2.1.17 and 2.1.19). The key difference is that the formula used to define the operator in Definitions 2.1.31 and 2.1.37 is a second-order formula. This gives rise to two new classes:

Definition 2.1.40. The logics SO(LFP) and SO(IFP) are respectively those that consist of least fixed-point and inflationary fixed-point operators, where the formula used to construct the relation is from second-order logic.

The results in this sub-section are drawn from Chapter 4 of [Imm99], Chapter 8 of [EF99] and [Kol07, Grä07], all of which contain further references.

2.1.2 Computational Complexity Theory

It is assumed that the reader is familiar with the ideas behind measuring the asymptotic growth of the resources required to compute the result of an algorithm for solving a problem [CLRS09]. Here, these ideas are used to present the main results of computational complexity theory [Pap94, Sip06].

A *problem* is a question asked about a particular class of structures. It is specified by the class of *input* structures it accepts, the *question* that it asks of some particular *input instance* and the type of *output* that will be produced as an answer to the question. The question is usually given in natural language although it can also be given as a logical specification. Problems are isomorphism-closed, in that they give the same answer for any pair of isomorphic structures. There are three main types of problems:

1. **Decision Problems.** The output of which is a boolean, e.g. yes/no or true/false.
2. **Optimisation Problems.** The output here is a quantity, usually a natural number.
3. **Construction Problems.** Also known as *function problems,* these give a structure as a result, often but not necessarily of the same type as the input structure.

Note that a decision problem is also an optimisation problem (but not necessarily vice versa) since the boolean output can be encoded as the numbers 0 and 1; also an optimisation problem is a construction problem (but not necessarily vice versa) since the output value is a type of structure. Throughout this thesis only decision and optimisation problems will be considered, although there will often be constructions as intermediates in calculating the answer.

An *algorithm* is a series of instructions that can be used to program a computer in order to work out the answer to a particular problem. By specifying an algorithm that computes a problem, an *upper bound* $O(f(n))$ for the complexity of solving that problem is immediately found, where n is the size of the input. The following definition classifies problems by the upper bound on their running times:

Definition 2.1.41. The class $\mathsf{DTIME}(f(n))$ contains all problems that have an algorithm that runs on a *deterministic computer* in time $O(f(n))$. The class $\mathsf{DSPACE}(f(n))$ contains all problems that have an algorithm that uses $O(f(n))$ space (memory) when run on a deterministic computer. The classes $\mathsf{NTIME}(f(n))$ and $\mathsf{NSPACE}(f(n))$ are similar except that the algorithm is run on a *non-deterministic computer.*

Here, the term *computer* is taken to mean any reasonable model of computation, i.e. one that is both well-defined and well-behaved. This generalisation of the term can be made since Church's Thesis states that no reasonable model of computation, such as the

Turing Machine or the Lambda Calculus, is considerably more powerful than any other. It should also be noted that to map an algorithm from one reasonable model of computation to another never requires more than a polynomial amount of time and a linear amount of space. Under this observation, problems can be classified independently from the models of computation that the algorithms run on as long as there is at most a polynomial difference in time and a linear difference in space between the models. This leads to the following (deterministic and non-deterministic) standard complexity classes for polynomials:

- $\mathsf{P} = \bigcup_{k\in\mathbb{N}} \mathsf{DTIME}\left(n^k\right)$
- $\mathsf{NP} = \bigcup_{k\in\mathbb{N}} \mathsf{NTIME}\left(n^k\right)$
- $\mathsf{PSPACE} = \bigcup_{k\in\mathbb{N}} \mathsf{DSPACE}\left(n^k\right)$
- $\mathsf{NPSPACE} = \bigcup_{k\in\mathbb{N}} \mathsf{NSPACE}\left(n^k\right)$

For logarithms:

- $\mathsf{L} = \mathsf{DSPACE}(\log n)$
- $\mathsf{NL} = \mathsf{NSPACE}(\log n)$

And for exponentials:

- $\mathsf{EXPTIME} = \bigcup_{k\in\mathbb{N}} \mathsf{DTIME}\left(2^{n^k}\right)$
- $\mathsf{NEXPTIME} = \bigcup_{k\in\mathbb{N}} \mathsf{NTIME}\left(2^{n^k}\right)$
- $\mathsf{EXPSPACE} = \bigcup_{k\in\mathbb{N}} \mathsf{DSPACE}\left(2^{n^k}\right)$
- $\mathsf{NEXPSPACE} = \bigcup_{k\in\mathbb{N}} \mathsf{NSPACE}\left(2^{n^k}\right)$

Since an algorithm that runs in $O(f(n))$ time (or space) also runs in $O(g(n))$ time (or space), where $O(g(n)) \geq O(f(n))$, it is the case that $\mathsf{DTIME}(f(n)) \subseteq \mathsf{DTIME}(g(n))$ for all $f(n)$ and $g(n)$ such that $O(g(n)) \geq O(f(n))$ and similarly for all the other classes, NTIME, DSPACE and NSPACE, in Definition 2.1.41, e.g. $\mathsf{P} \subseteq \mathsf{EXPTIME}$. In time $O(f(n))$ an algorithm can only use $O(f(n))$ space, which leads to the relationships between time and space $\mathsf{DTIME}(f(n)) \subseteq \mathsf{DSPACE}(f(n))$ for deterministic complexity classes and $\mathsf{NTIME}(f(n)) \subseteq \mathsf{NSPACE}(f(n))$ for non-deterministic ones.

The containments in this hierarchy for deterministic computers are strict for time when $g(n)$ is at least the square of $f(n)$ and space when $g(n)$ is at least a factor $(\log n)$ larger than $f(n)$; a similar result holds for non-deterministic computers. These results are collectively know as the *Space/Time Hierarchy*:

Theorem 2.1.42 (Time and Space Hierarchy)**.** *For deterministic classes of problems, the following strict hierarchies hold:*

- $\mathsf{DTIME}(f(n)) \subset \mathsf{DTIME}(f^2(n))$
- $\mathsf{DSPACE}(f(n)) \subset \mathsf{DSPACE}(f(n).\log n)$

and for non-deterministic classes of problems the containments are:

- $\mathsf{NTIME}(f(n)) \subset \mathsf{NTIME}(f(n).\log n)$
- $\mathsf{NSPACE}(f(n)) \subset \mathsf{NSPACE}(f(n).\log n)$

From the Time and Space Hierarchy Theorem the following containments for the standard complexity classes can be derived:

Corollary 2.1.43. *The containments:*

1. $\mathsf{P} \subset \mathsf{EXPTIME}$
2. $\mathsf{NP} \subset \mathsf{NEXPTIME}$
3. $\mathsf{L} \subset \mathsf{PSPACE} \subset \mathsf{EXPSPACE}$
4. $\mathsf{NL} \subset \mathsf{NPSPACE} \subset \mathsf{NEXPSPACE}$

are all direct from Theorem 2.1.42.

The Time and Space Hierarchy Theorem only details the relationship between different functions within a type of complexity class; what is doesn't do is describe relations between deterministic and non-deterministic complexity classes or space and time complexity classes. The arguments that allow these different types of complexity classes to be related are based on *simulation*; that is, given an algorithm that runs on one type of computer using a certain amount of time or space another algorithm can be described that simulates the first algorithm on a different type of computer in a related amount of time or space.

A non-deterministic computer can simulate a deterministic computer (since every deterministic program is also a non-deterministic one, but not necessarily vice versa) in an equal amount of time or space and so it is the case that for any $f(n)$, $\mathsf{DTIME}(f(n)) \subseteq \mathsf{NTIME}(f(n))$ and $\mathsf{DSPACE}(f(n)) \subseteq \mathsf{NSPACE}(f(n))$ so, for example, $\mathsf{P} \subseteq \mathsf{NP}$.

Any deterministic computer can simulate a non-deterministic computer by going through every possible computation path sequentially. Whilst this method of simulation clearly requires more time, as there are a possibly exponential number of computation paths, it does not require any additional space, since each path of computation can be considered independently from all others. Since this simulation of the non-deterministic computer requires the same degree of space, it is the case that $\mathsf{NTIME}(f(n)) \subseteq \mathsf{DSPACE}(f(n))$.

With the relationships between complexity classes in the previous paragraphs it is possible to construct a non-strict hierarchy of classes for some function $f(n)$ up to $\mathsf{NSPACE}(f(n))$. To go beyond this a relationship between $\mathsf{NSPACE}(f(n))$ and a complexity class based on an asymptotically larger function $g(n)$ is required, since as has just been shown, NSPACE is the largest of the four complexity classes for some fixed function $f(n)$. As previously stated, a deterministic machine can simulate a non-deterministic one by sequentially executing every possible path of computation. Since there are an exponential number of paths a deterministic computer can simulate a non-deterministic one in $2^{f(n)}$ steps. The relationships here is that $\mathsf{NSPACE}(f(n)) \subseteq \bigcup_{k\geq 1} \mathsf{DTIME}\left(2^{k.f(n)}\right)$ (see Chapter 7 of [Pap94] for more details).

Another relationship for non-deterministic space is with deterministic space and is given by the following theorem

Theorem 2.1.44 (Savitch's Theorem [Sav70])**.** $\mathsf{NSPACE}(f(n)) \subseteq \mathsf{DSPACE}(f^2(n))$

An immediate consequence of this theorem is that for functions that grow at a polynomial or higher rate, non-deterministic space classes collapse to their deterministic versions and in particular:

Corollary 2.1.45. $\mathsf{PSPACE} = \mathsf{NPSPACE}$ *(and* $\mathsf{EXPSPACE} = \mathsf{NEXPSPACE}$*)*

From the above relationships between classes, a hierarchy of the standard complexity classes can be used to constructed:

$$\mathsf{L} \subseteq \mathsf{NL} \subseteq \mathsf{P} \subseteq \mathsf{NP} \subseteq \mathsf{PSPACE} \subseteq \mathsf{EXPTIME} \subseteq \mathsf{NEXPTIME} \subseteq \mathsf{EXPSPACE} \tag{2.1}$$

As is regularly stated, it is surprising that very little is know about the relationships between these classes. The only known proper inclusions are those due to the Time and Space Hierarchy Theorem (2.1.42) that are stated in Corollary 2.1.43 plus one further proper containment resulting from the Corollary succeeding Savitch's Theorem (2.1.44).

Corollary 2.1.46. $\mathsf{NL} \subset \mathsf{PSPACE}$

2.1.2.1 Complement of complexity classes

Every problem Q has a complementary problem $\overline{Q}$ whose output for a particular input structure is the complement of the output of Q. In the realm of decision problems, the complement of the output *yes* is the output *no* and vice versa.

Definition 2.1.47. The *complement* of a decision problem Q is the decision problem $\overline{Q}$ that is defined as follows: Given any input structure $\mathcal{A}$ appropriate to Q, $Q(\mathcal{A}) = yes$ iff $\overline{Q}(\mathcal{A}) = no$ and *vice versa.* That is, the complement problem answers *yes* when the original problem answers *no*, and similarly the complement problem answers *no* when the original problem answers *yes*; this relationship holds for any valid input structure.

The complement of a complexity class is the set of all the complements of problems from the original class. For example the class $\mathsf{co\text{-}P}$ contains all the problems that are a complement of some problem in P. An interesting property of complexity classes is whether or not they are *closed under complement,* which simply put asks whether the complement of a class is equal to the original class.

Clearly, any deterministic class is closed under complement, as the deterministic computer simply executes the algorithm and switches the output from either *yes* to *no* or *no* to *yes*, which does not increase the complexity of solving the problem.

In the case of non-deterministic complexity classes, it is not obvious how to write an algorithm that takes an algorithm for a problem and computes the complement of that problem, whilst still remaining in the same complexity class. To see this, recall that a non-deterministic computer returns *yes* if there *exists at least one* sequence of computational steps that leads to an accepting state and *no* if *every* sequence of computational steps ends up in a *failed* state; that is one where the algorithm contains no rules for how to move on to a new state. This asymmetry between the way that a non-deterministic computer arrives

at a *yes* or a *no* answer means that simply running the algorithm and then negating the result cannot work, as the algorithm answers *no* once every computational path has been tried and all have failed. Once in a failed state the non-deterministic computer cannot, by definition, move to an accepting state.

For time classes, the question as to whether they are closed under complement or not is a major open question in the field of computational complexity (see Chapter 10 of [Pap94]). But for space classes, the question has been answered by the following theorem:

Theorem 2.1.48 (The Immerman-Szelepscényi Theorem [Imm88, Sze87])**.** *If* $f \geq \log n$ *then* $\mathsf{NSPACE}(f(n)) = \mathsf{co\text{-}NSPACE}(f(n))$.

2.1.2.2 Reductions and Completeness

So far, the computational complexity of a problem has only been discussed with respect to which complexity classes it belongs to. In this subsection, an important tool for relating problems to each other within a complexity class, called *reduction,* is introduced and from this the concept of problems that are *complete* for a complexity class is arrived at. Conceptually, complete problems are very important in understanding the nature of a complexity class since they are the *hardest* problems in the class and as such encapsulate, through reductions, what it means to be a member of that class.

A reduction is a transformation of the input of some problem P into the input of some other problem Q in such a way that an algorithm that solves Q can be applied to a transformed input instance of P in order to solve P. More formally,

Definition 2.1.49. A (many-to-one) reduction from the problem P to the problem Q consists of a transformation function $T : \mathcal{A} \mapsto \mathcal{B}$, where $\mathcal{A}$ is an input instance to P and $\mathcal{B}$ is an input instance to Q. As T is total, it must provide a mapping from every valid instance of P to a valid instance of Q. It must hold that for any input structure $\mathcal{A}$ (appropriate to problem P), if it is a *yes* instance of P then $T(\mathcal{A})$ must also be a *yes* instance of Q; conversely, if $\mathcal{A}$ is a *no* instance of P then $T(\mathcal{A})$ is a *no* instance of Q. When such a transformation between two problems P and Q exists, it is said that there is a reduction from P to Q, written $P \leq Q$.

It is very important to analyse the complexity of computing the transformation function in a reduction, especially when reductions are used to order the complexity of problems *within a complexity class.* The hardest problems in the class NP are known as NP-complete problems.

Definition 2.1.50. A problem Q is NP-complete iff $Q \in \mathsf{NP}$ and for every problem $P \in \mathsf{NP}$ there exists a reduction (see Definition 2.1.49) computable in polynomial time so that $P \leq Q$; i.e. there exists a polynomial-time many-to-one reduction from P to Q.

The first problem discovered to be NP-complete was Satisfiability, or Sat, (see Definition 2.1.59) by Cook [Coo71]. Once Sat was shown to be NP-complete many other problems in NP were also shown to be NP-complete by giving polynomial-time many-to-one reductions to them from Sat; the Appendix of [GJ79] contains a large but by no means complete catalogue of NP-complete problems.

When ordering the complexity of problems within P a polynomial-time many-to-one reduction is useless because P is closed under such reductions; any problem in P can be trivially transformed to any other problem in P in polynomial-time, since this is the amount of time required to calculate a solution. For this reason *log-space* reductions are used, which are weaker than polynomial-time reductions under the assumption that $\mathsf{L} \neq \mathsf{P}$.

Definition 2.1.51. A problem Q is P-complete iff $Q \in \mathsf{P}$ and for every problem $R \in \mathsf{P}$ there exists a log-space reduction from R to Q.

There are many problems that have been shown to be complete for P; a compendium of such problems can be found in Appendix A of Part II of [GHR95].

Another class of problems is that of the problems that are at least as hard as the hardest problems in NP (see Chapter 5 of [GJ79] for more information on NP-hardness). These use a different type of reduction from that presented in Definition 2.1.49.

Definition 2.1.52. A polynomial-time *Turing Reduction* from P to Q consists of an oracle for solving problem Q and an algorithm that queries the oracle and produces an answer to instances of problem P in polynomial-time.

The polynomial-time Turing Reduction differs from the many-to-one reduction in that it can apply the oracle for solving the problem Q multiple times, rather than just once. Whilst this ability is not required for NP-complete and P-complete reductions it is required for NP-hard reductions:

Definition 2.1.53. A problem Q is NP-hard iff there exists a polynomial-time Turing reduction from some NP-complete problem P, to Q.

Note that while P must be a decision problem, Q does not necessarily have to be; it could for example, be an optimisation problem. Definitions 2.1.50 and 2.1.53 give rise to the following relationship within (and beyond) NP:

$$\mathsf{NP} \leq \mathsf{NP}\text{-complete} \leq \mathsf{NP}\text{-hard}$$

In Chapter 3 the class NP-hard is used extensively since it contains the hardest optimisation problems whose decision variants are in NP.

2.1.3 Satisfiability Problems

There is one large class of computational problems that deserve special mention as they often crop up when dealing with reductions, completeness and logic. These are the *satisfiability* problems and their derivatives. Here, they are presented in the context of finite model theory.

Definition 2.1.54. Let φ be a first-order, quantifier-free, Boolean formula over the set of variables $V = \{x_1, ..., x_n\}$ (i.e. $\text{free}(\varphi) = V$ or equivalently φ is appropriate to the vocabulary $\langle x_1, ..., x_n \rangle$). Let the unary relation T be a *truth assignment* to the variables in V; that is, T is a $\langle x_1, ..., x_n \rangle$ structure. If there exists a T such that $T \models \varphi$ then φ is *satisfiable*; if $T \models \varphi$ for all possible truth assignments then φ is a *tautology* or is *valid*, often simply written $\models \varphi$; if $T \not\models \varphi$ for all possible truth assignments then φ is *unsatisfiable* or *invalid*, often simply written $\not\models \varphi$.

Furthermore, it is desirable to work with Boolean formulae in a normal form:

Definition 2.1.55. A Boolean formula is in *conjunctive normal form (CNF)* if and only if it consists of a conjunction of *clauses* that themselves are disjunctions of *literals*.

The terms *clause* and *literal* from the above definition are commonly used when talking about the components of a Boolean formula and so deserve some further explanation.

- A *literal* is a Boolean variable from the set of variables V proceeded either with or without a negation symbol: x_1 and $\neg x_1$ are both literals. x_1 is called a *positive* literal and $\neg x_1$ is called a *negative* literal.
- A *clause* is a fragment of a Boolean formula that consists only of literals and the *or* operator; it is a disjunction of literals: $x_1 \vee \neg x_2 \vee x_3$ is a clause of three literals.

So a formula in conjunctive normal form is a conjunction of clauses: e.g. $(x_1 \vee \neg x_2) \wedge x_3$ is an example of a Boolean formula in CNF.

Remark 2.1.56. In some literature the notation $\bar{x}_1$ is used to denote the negation of the variable x_1. In order to avoid confusion the negation operator is used in this thesis, as the notation $\bar{x}$ (read: x-bar) is used to denote a tuple.

Remark 2.1.57. A literal can technically appear in a clause more than once, although it can be safely assumed that this doesn't happen, as the repetition of a literal can be removed, e.g. $x_1 \vee x_2 \vee x_2 \equiv x_1 \vee x_2$. If a variable appears both positively and negatively in a clause then the clause itself can be assumed true and removed, since $\models (x_1 \vee \neg x_1)$.

In the context of finite model theory, the problem of satisfiability is usually represented as follows:

Definition 2.1.58. The vocabulary of the input structure to a satisfiability problem is $\sigma_{SAT} = \langle P^2, N^2 \rangle$. The structure $\mathcal{A}$ realises the vocabulary σ_{SAT} such that $P(x_i, c_i)$ is true iff variable x_i appears positively in clause c_i and $N(x_i, c_i)$ is true iff variable x_i appears negatively in clause c_i. Note that this allows for at most $|A|$ clauses and $|A|$ variables to be represented by $\mathcal{A}$.

Now the main computational problem is given:

Definition 2.1.59. The problem SATISFIABILITY (or SAT for short) has an input structure $\mathcal{A}$ as per Definition 2.1.58 and asks the question: is there a truth assignment T to the Boolean variables $x_1, ..., x_n$, such that the Boolean formula represented by the structure $\mathcal{A}$ is satisfied?

The whole of the theory of NP-completeness (see Section 2.1.2.2 and Definition 2.1.50) stems from the following result:

Theorem 2.1.60 (Cook's Theorem [Coo71])**.** SAT *is NP-complete.*

and

Theorem 2.1.61 ([Coo71])**.** 3-SAT*, the restriction of* SAT *to a Boolean formulae whose clauses have at most 3 literals, is also NP-complete.*

Many other variations of SAT are also NP-complete; Appendix 9 of [GJ79] lists more such variations.

2.1.3.1 Horn Clauses

There is another interesting structural restriction to clauses that greatly effects the complexity of the corresponding satisfiability problem; clauses obeying this restriction are called *Horn clauses.*

Definition 2.1.62. A Horn clause is a disjunction of literals in which only one literal is positive.

For example, the clause $x_1 \vee \neg x_2$ is Horn but the clause $x_1 \vee x_2$ is not. In general, a Horn clause of the form $x_1 \vee \neg x_2 \vee ... \vee \neg x_n$ can be written as $(x_2 \wedge ... \wedge x_n) \Rightarrow x_1$, due to the logical equivalences $u \Rightarrow v \equiv \neg u \vee v$ and $\neg(\neg u \vee \neg v) \equiv u \wedge v$. This form involving implication is commonly used when writing out Horn formula as is highlights the key feature of a Horn clause: if all the negative literals (those on the left-hand side of the implication) are true, then the positive literal (the one on the right-hand side of the implication) must be true.

Definition 2.1.63. The problem Horn-Sat is a restriction of Sat (see Definition 2.1.59) in which every clause is a Horn clause (see Definition 2.1.62).

Taking a Horn-Sat instance to have the same vocabulary and structure as that of a Sat instance, i.e. as in Definition 2.1.58, the formula represented by such a structure $\mathcal{A}$ is in Horn form iff

$$\mathcal{A} \models \forall c \forall x \forall y (P(x, c) \wedge P(y, c) \Rightarrow x = y)$$

and hence such a structure represents a formula consisting of a conjunction of Horn clauses. This is true because if any pair of variables x and y appear in the same clause positively (i.e. $P(x, c)$ and $P(y, c)$ are true for all clauses c) then it must be the case that x and y are the same variable, otherwise the Boolean formula represented by the structure is not Horn.

This restriction of a formula to Horn clauses makes the satisfiability problem significantly easier to solve than in the general case (see Theorem 2.1.60).

Theorem 2.1.64. Horn-SAT *is* P*-complete.*

This result comes from problem A.6.3 in the compendium of P-complete problems [GHR95]; it is attributed to [Pla84].

In the context of finite model theory, a formula can be in Horn form if it has certain properties:

Definition 2.1.65. A first-order formula ψ is said to be *Horn with respect to a vocabulary* σ iff ψ is in CNF and each clause has a most one positive atom involving a relation symbol from σ.

There are various classes of logic that are defined as being Horn. The most common is second-order Horn logic, which contains formulae that are Horn with respect to the second-order relational variables they contain:

Definition 2.1.66. The logic *second-order Horn* (denoted SO-Horn) consists of formulae of the form:

$$\exists S_1, ...\exists S_\alpha \forall x_1, ..., \forall x_\beta \, \psi$$

where ψ is a first-order quantifier free formula appropriate to some vocabulary $\sigma \cup \tau$; $\tau = \langle S_1, ..., S_\alpha \rangle$ is a vocabulary of relation symbols, each of some given arity and $x_1, ..., x_\beta$ are first-order variables. Moreover, the formula ψ is Horn with respect to the vocabulary τ.

Remark 2.1.67. This logic is called SO-Horn as opposed to $\exists$SO-Π_1-Horn as it has been shown that all families of second-order Horn logic collapse to the above class [Grä91b].

2.1.3.2 Krom Clauses

A Krom clause (like a Horn clause) is specified by placing a restriction on clauses in a formula; a formula that consists entirely of Krom clauses is said to be a Krom formula.

Definition 2.1.68. A *Krom clause* is a disjunction that contains at most two distinct literals.

The restriction of SAT to Krom formulae is surprisingly called 2-SAT rather than KROM-SAT. It is even easier to solve than HORN-SAT.

Theorem 2.1.69. 2-SAT, *the restriction of* SAT *to Boolean formulae whose clauses are Krom clauses, is* NL*-complete.*

This result is attributed by [Imm99] to [JLL76].

2.1.3.3 Optimisation Variants of Satisfiability Problems

Satisfiability problems are generalised into optimisation problems by adding some constant k and asking if there exists a truth assignment that satisfies at least k clauses in the Boolean formula. The maximisation version of SAT is given (as a decision problem) as:

Definition 2.1.70. The decision problem MAX-SAT (short for *Maximum Satisfiability*) is given an input structure $\mathcal{A}$ of vocabulary σ_{SAT} (see Definition 2.1.58) plus an integer constant k and asks the question: is there a truth assignment T to the variables in $\mathcal{A}$ such that the number of satisfied clauses $|C| \geq k$, where C is defined as:

$$C := \{c \in A \mid \exists x((P(x,c) \wedge T(x)) \vee (N(x,c) \wedge \neg T(x))\}$$

The usual restrictions to SAT, e.g. restricting the number of literals in each clause can also be applied to MAX-SAT, resulting in MAX-2-SAT, MAX-3-SAT etc. Similarly, the type of clauses can be restricted, e.g. to Horn clauses, resulting in MAX-HORN-SAT. Both these restrictions can be combined, resulting in say, MAX-2-HORN-SAT.

The results in this subsection are covered in depth in books on algorithms, e.g. [CLRS09, Pap94]; the relationship between SAT and NP-completeness is well studied in [GJ79]; a good survey of propositional logic (i.e. Boolean logic) problems and algorithms can be found in [BL99].

2.1.4 Descriptive Complexity

The area of study concerned with finding the relationships between computational complexity classes and classes of logical formulae is called *Descriptive Complexity*.

2.1.4.1 Characterising a class of structures using a sentence (both stable and unstable)

Recall that the set of all structures for some relational vocabulary τ is denoted STRUCT[τ] and contains all the finite τ-structures. Now let $K \subseteq$ STRUCT[τ] be an isomorphism-closed class of τ-structures. A formula φ of some logic $\mathcal{L}$ is said to *characterise* the class K if it can distinguish whether a τ-structure is a member of K or not.

Definition 2.1.71. The class K of τ-structures is characterised by the formula φ of some logic $\mathcal{L}$ iff for any τ-structure $\mathcal{A}$ the following conditions hold:

i) $\mathcal{A} \in K \Leftrightarrow \mathcal{A} \models \varphi$

ii) $\mathcal{A} \notin K \Leftrightarrow \mathcal{A} \not\models \varphi$

Now, when STRUCT[τ] coincides with the input instances to a problem $\mathcal{Q}$ it is possible to construct a subclass of τ-structures $K_{\mathcal{Q}}$ such that $\mathcal{A} \in$ STRUCT[τ] and $\mathcal{A} \in K_{\mathcal{Q}}$ iff $\mathcal{A}$ is a *yes*-instance to problem $\mathcal{Q}$. Any formula that characterises the class of structures $K_{\mathcal{Q}}$ is therefore a formula that characterises the problem $\mathcal{Q}$.

To see this more clearly, some examples are given below of how to characterise problems on graphs using sentences of first-order logic.

Example 2.1.72. Given some digraph $\mathcal{G}$ of the vocabulary τ_G,

$$\mathcal{G} \models \forall x \forall y (E(x, y) \Rightarrow E(y, x))$$

iff the digraph $\mathcal{G}$ is undirected and is, in fact, a graph.

Example 2.1.73. Given some (di-)graph $\mathcal{G}$ of the vocabulary τ_G,

$$\mathcal{G} \models \neg \exists x \exists y \exists z (E(x, y) \wedge E(y, z) \wedge E(z, x))$$

iff the (di-)graph $\mathcal{G}$ is triangle free.

It is assumed that the input structure of any problem can be encoded as a relational structure of some vocabulary that is fixed for the problem. This is a fair assumption since a model of computation, such as a Turing machine, encodes its input as a binary string, which can be encoded as a τ_s-structure; with $\tau_s = \langle \leq, S^1 \rangle$, where $\leq$ is a total ordering of the universe and the unary relation S indicates the positions in the string that are set to 1.

In the examples above, the formulae given are both sentences and are referred to as *stable* characterisations of the given problem. If the formula is parameterised, that is the exact formula depends on the input structure in question, then it is referred to as an *unstable* characterisation of the given problem.

2.1.4.2 Characterising a complexity class using a class of logic

The real power of Descriptive Complexity comes from its ability to equate a complexity class to a class of logical formulae.

Definition 2.1.74. Some logic $\mathcal{L}$ characterises some complexity class C under the following conditions:

(i) For every problem $\mathcal{Q} \in C$, there exists a formula $\varphi \in \mathcal{L}$ such that φ characterises $\mathcal{Q}$.

(ii) Every formula $\varphi \in \mathcal{L}$ characterises a problem in C.

When $\mathcal{L}$ characterises C, it is written that $\mathcal{L} = C$.

Whilst not strictly the case, all the characterisations of the key complexity classes are by formulae that are stable and so unless otherwise stated it is assumed that only logics containing stable formulae are being considered.

2.1.4.3 Characterisations of Complexity Classes

The first characterisation of a complexity class by a logic is due to Fagin's seminal work in the field. It shows that a class of second order logic (see Definition 2.1.22) characterises non-deterministic, polynomial-time decision problems.

Theorem 2.1.75 (Fagin's Theorem [Fag74])**.** *A problem is in* NP *if, and only if, it can be characterised by a sentence of existential second-order logic* ($\exists$SO)*.*

Interestingly, the dual of $\exists$SO, *universal second-order logic* ($\forall$SO) characterises the complement of NP: co-NP = $\forall$SO. Second-order logic in general characterises the polynomial-time hierarchy. When restricted to formulae that are Horn with respect to the relational variables in a formula, the resulting logic SO-Horn (see Definition 2.1.66) gives the following result:

Theorem 2.1.76 (Grädel's Theorem [Grä91a, Grä91b, Grä92])**.** *A problem is in* P *if, and only if, it can be defined by a sentence of* SO-Horn *in the presence of a built-in successor relation.*

Away from second-order logic, first-order logic augmented with various operators provides several characterisations. It should be noted though that first-order logic on its own is too weak to characterise P (see Chapter 2 of [EF99]).

One such characterisation using augmented first-order logic is independently due to Immerman and Vardi and uses the least fixed-point operator (see Definition 2.1.34).

Theorem 2.1.77 (Immerman [Imm86] & Vardi [Var82])**.** *The logic* FO(LFP) *characterises the complexity class* P *on ordered structures.*

The problem with FO(LFP) is that it requires the formulae that define operators to only positively refer to the relation they are constructing. This syntactic restriction is inconvenient when describing problems but handily the less syntactically restrictive inflationary fixed point operator (see Definition 2.1.37) is expressively equivalent to LFP:

Theorem 2.1.78 (Gurevich-Shelah [GS86])**.** IFP = LFP *on finite structures.*

The application of the equivalence in Theorem 2.1.78 to the result in Theorem 2.1.77 gives the following Corollary:

Corollary 2.1.79. *The logic* FO(IFP) *characterises the complexity class* P *on ordered structures.*

The logical equivalence defined in Theorem 2.1.36 that says that there is no expressive hierarchy in nested LFP or IFP operators, means that any sentence in either FO(IFP) or FO(LFP) that captures a problem from P is logically equivalent to another sentence in the same logic that only has one fixed point operator.

Remark 2.1.80. The logics that characterise P in Theorem 2.1.77 and Corollary 2.1.79 only do so on ordered structures; that is, structures that have an ordered universe through the inclusion of a built in ordering relation (see Definition 2.1.26). In the case of SO-Horn though (see Theorem 2.1.76), only a *successor relation* is required. It is an open question as to whether or not a logic exists that can characterise P without the addition of a built-in ordering relation.

2.1.5 Games and Inexpressibility

In showing that a particular problem can be expressed using a sentence of some logic (see Section 2.1.4.1 for details of what is meant by this), in essence all that has been done is that an upper bound for the problem has been given; the same as would happen when demonstrating an algorithm for solving the problem. The main advantage of expressing a problem using logic, given by Descriptive Complexity, is that the complexity class that the problem is in follows immediately, without any analysis of the algorithm.

What is more interesting, as was mentioned in the Introduction, is what a problem's *lower bounds* are. For example, given an algorithm that solves the problem Q in polynomial time, this shows that $Q \in \mathsf{P}$ and gives an upper bound. What would be more interesting is if it could be shown that Q's lower bound is also in P; that is, there is no algorithm for Q that would place its upper bound in NL. The reason this is interesting is that it would show that NL is strictly contained within P; a containment that is currently unknown (see the complexity class hierarchy in Equation 2.1 on page 26 and the known containments in Corollary 2.1.43).

2.1.5.1 Inexpressibility in Logic

In the context of general computational models, such as the Turing Machine, proving a lower bounds for a problem is a difficult task, since every single possible programming of the machine must be considered and analysed. One of the great hopes in proving lower bounds is finite model theory, due to its characterisations of the key complexity classes and existing tools/methods from mathematical model theory.

In a finite model theory context a lower bound is obtained by demonstrating that no formula in a certain logic can characterise a particular problem. One method for doing this, using logical equivalence of structures, is now given:

Definition 2.1.81. Given a class of τ-structures K and a structural equivalence relation for some logic $\mathcal{L}$ (see Definition 2.1.13), it is said that K is $\mathcal{L}$-inexpressible if, and only if,

there exists two τ-structures $\mathcal{A} \in K$ and $\mathcal{B} \notin K$ such that $\mathcal{A} \equiv_{\mathcal{L}} \mathcal{B}$, that is $\mathcal{A}$ and $\mathcal{B}$ are $\mathcal{L}$-equivalent.

Using this method of showing inexpressibility in a logic gives rise to another method for showing strict hierarchies between logics:

Proposition 2.1.82. *Given a class of τ-structures K and two logics $\mathcal{L}_1$ and $\mathcal{L}_2$ such that $\mathcal{L}_1 \subseteq \mathcal{L}_2$, if K is $\mathcal{L}_2$-expressible (that is, there exists some $\varphi \in \mathcal{L}_2$ such that φ characterises K) and K is $\mathcal{L}_1$-inexpressible then it follows that $\mathcal{L}_1 \subset \mathcal{L}_2$, i.e. the containment between the logics is proper.*

Now, let K be the set of τ-structures that represent yes-instances of the problem Q; let $\mathcal{L}_1$ be a logic that characterises the complexity class P; let $\mathcal{L}_2$ be a logic that characterises the complexity class NP and using the method from Proposition 2.1.82 show that $\mathcal{L}_1 \subset \mathcal{L}_2$, which would yield the result $\mathsf{P} \subset \mathsf{NP}$.

The main difficulties with this method, aside from choosing K, $\mathcal{A}$ and $\mathcal{B}$, are proving that $\mathcal{A} \equiv_{\mathcal{L}} \mathcal{B}$.

2.1.5.2 The Ehrenfeucht-Fraïssé Game

A method of showing structural equivalence in first-order logic was developed independently by Ehrenfeucht [Ehr61] and Fraïssé [Fra54], with the resulting method now commonly referred to as the *Ehrenfeucht-Fraïssé Game* [EF99, Lib04, GKL$^+$07].

Definition 2.1.83 (Ehrenfeucht-Fraïssé Game)**.** The game $G_r(\mathcal{A}, \mathcal{B})$ is played on two τ-structures $\mathcal{A}$ and $\mathcal{B}$ by two players called *Spoiler* and *Duplicator* (sometimes referred to as *Player I* and *Player II*, or *Samson* and *Delilah*). A play of the game consists of r rounds. In each round the Spoiler chooses a structure, either $\mathcal{A}$ or $\mathcal{B}$, and places a pebble on a point in that structure (i.e. a member of the universe); the Duplicator then responds by placing a pebble on a point in the other structure. After the i^{th} round the pebble a_i sits on structure $\mathcal{A}$ and b_i on structure $\mathcal{B}$: this tuple (a_i, b_i) is the *play* for that round.

Once all r rounds have been played there will be r tuples resulting from r plays. Two sub-structures $\mathcal{A}'$ and $\mathcal{B}'$ are then constructed by restricting $\mathcal{A}$ and $\mathcal{B}$ to the members of their universes that have pebbles placed on them. A mapping $p : \mathcal{A}' \to \mathcal{B}'$ is then constructed such that for each $0 < i \leq r$, $p(a_i) = b_i$. The Duplicator is the winner of the game if p describes a valid isomorphism between $\mathcal{A}'$ and $\mathcal{B}'$, otherwise the Spoiler is the winner of the game.

In effect the Spoiler is aiming to show that the two structures can be differentiated between by the logic whereas the Duplicator's aim is to stop the Spoiler from being able to do this, hence showing that the logic cannot differentiate between them. The Duplicator is said to have a *winning strategy* if she can always respond to Spoiler's moves and maintain an isomorphism between the pebbled sub-structures, whereas the Spoiler is said to have a *winning strategy* if he can always move in any sequence of plays such that Duplicator cannot respond in a way that maintains an isomorphism between the pebbled sub-structures. It is just said that the Spoiler or the Duplicator *wins* the game if they have a winning strategy.

Remark 2.1.84. The Ehrenfeucht-Fraïssé game was originally designed to work with infinite structures, on which games with an infinite number of rounds would be played. This poses

a problem on finite structures in that after an infinite number of moves every point in the universes of the structures will be pebbled, reducing the game to the simpler concept of isomorphism. To get around this and indeed to make the game useful, the number of rounds of play is introduced as a parameter to the game and with this, the equivalence of structures in a parameterised restriction of first order logic can be examined.

The relationship between the game and equivalence of structures in first-order logic is given by the following theorem:

Theorem 2.1.85. *For two τ-structures $\mathcal{A}$ and $\mathcal{B}$ the following are equivalent:*

(i) $\mathcal{A} \equiv_{\mathrm{FO}_r} \mathcal{B}$.

(ii) Duplicator wins the game $G_r(\mathcal{A}, \mathcal{B})$.

where FO_r *is the class of first-order logic formulae restricted to a quantifier depth of at most* r *(see Definition 2.1.15).*

Unfortunately this result is slightly different from the inexpressibility required by the method in Proposition 2.1.82 as the Ehrenfeucht-Fraïssé game, in its above stated form, can only be used to show that a problem is FO_r-inexpressible for some fixed r, not FO-inexpressible. The way around this is as follows:

Proposition 2.1.86. *For a class of τ-structures K, the following are equivalent:*

i) K *is* FO*-inexpressible.*

ii) For all $r \geq 1$*, there exists two structures* $\mathcal{A} \in K$ *and* $\mathcal{B} \notin K$ *such that* $\mathcal{A} \equiv_{\mathrm{FO}_r} \mathcal{B}$.

iii) For all $r \geq 1$*, there exists two structures* $\mathcal{A} \in K$ *and* $\mathcal{B} \notin K$ *such that Duplicator wins* $G_r(\mathcal{A}, \mathcal{B})$.

Whilst this method must be considered for all r and so leads to an infinite number of cases, it is usual to describe each structure as being parameterised by r. Consider for example the case when the vocabulary τ is the vocabulary of graphs τ_g (see Definition 2.1.5): $\mathcal{A}$ could be an undirected cycle of length r and $\mathcal{B}$ could be a undirected cycle of length $r+1$ with the winning strategy for the Duplicator being proven by induction on r.

Unfortunately, since first-order logic does not characterise any of the major complexity classes (see the list in Section 2.1.4.3) the application of the method presented in Proposition 2.1.82 using Proposition 2.1.86 will not lead to showing any of the containments in the hierarchy from Equation 2.1 on page 26 to be strict; at best it could show strict containments between fragments of first-order logic.

2.1.5.3 Extending Ehrenfeucht-Fraïssé Games to Second-Order Logic

The game described in Definition 2.1.83 can be extended to second-order logic [Fag75].

Definition 2.1.87. The second-order Ehrenfeucht-Fraïssé game is, like the first-order one in Definition 2.1.83, played on two τ-structures $\mathcal{A}$ and $\mathcal{B}$ for r-rounds. It is also parameterised by a sequence of arities $\bar{s} = \langle s_1, ..., s_p \rangle$. Initially, the Spoiler chooses a structure (either $\mathcal{A}$ or $\mathcal{B}$) and for each arity s_i picks some relation $\mathcal{C}_i \subseteq A^{s_i}$ if they chose

$\mathcal{A}$, or $\mathcal{D}_i \subseteq B^{s_i}$ if they chose $\mathcal{B}$. The Duplicator then responds by picking $\mathcal{D}_i \subseteq B^i$ for each arity s_i (if Spoiler chose $\mathcal{A}$) or $\mathcal{C}_i \subseteq A^i$ for each arity s_i (if Spoiler chose $\mathcal{B}$). The game then continues for r rounds on the extended structures $(\mathcal{A}, \mathcal{C}_1, ..., \mathcal{C}_p)$ and $(\mathcal{B}, \mathcal{D}_1, ..., \mathcal{D}_p)$ in exactly the same way as the first-order Ehrenfeucht-Fraïssé game, i.e. the winner is the winner of the game $G_r((\mathcal{A}, \mathcal{C}_1, ..., \mathcal{C}_p), (\mathcal{B}, \mathcal{D}_1, ..., \mathcal{D}_p))$.

The game is denoted $G_{\langle s_1,...,s_p\rangle,r}(\mathcal{A}, \mathcal{B})$ and two τ-structures $\mathcal{A}$ and $\mathcal{B}$ are $\mathrm{SO}_{\langle s_1,...,s_p\rangle,r}$-equivalent if, and only if, the Duplicator wins $G_{\langle s_1,...,s_p\rangle,r}(\mathcal{A}, \mathcal{B})$, where the logic $\mathrm{SO}_{\langle s_1,...,s_p\rangle,r}$ is the fragment of SO in which the p second-order quantifiers each of arity s_i appear followed by a first-order formula from the fragment FO_r. When the arities are all 1, that is each relation that is picked is unary, then the game is denoted as simply $G_{p,r}(\mathcal{A}, \mathcal{B})$.

To extend the method of showing inexpressibility of a problem in first-order logic to second-order logic using the above game, would require structures to be found for every possible combination of arities for every $p \geq 1$ as well as for every $r \geq 1$. This makes the problem of finding structures much harder and to date, only results that show inexpressibility in fragments of second-order logic that are restricted so that each $s_i = 1$ (i.e. Monadic Second-order logic or MSO) have been found [Fag75, dR87]. So far, inexpressibility results with even just $s_1 = 2$ have not been found [Fag96].

2.1.5.4 The Ajtai-Fagin Game

In order to progress with developing winning strategies for the Duplicator on second-order Ehrenfeucht-Fraïssé games the game itself needed to be made easier for the Duplicator to win. One such restriction was given by Ajtai and Fagin:

Definition 2.1.88 (Ajtai-Fagin Game [AF90])**.** The Ajtai-Fagin game is the same as the second-order Ehrenfeucht-Fraïssé game, except that a class of τ-structures K is provided as input rather than two τ-structures $\mathcal{A}$ and $\mathcal{B}$. The play proceeds as follows:

(i) Duplicator picks a structure $\mathcal{A} \in K$.

(ii) For each arity s_i the Spoiler picks a relation $\mathcal{C}_i \subseteq A^{s_i}$.

(iii) Duplicator picks a structure $\mathcal{B} \notin K$.

(iv) For each arity s_i the Duplicator picks a relation $\mathcal{D}_i \subseteq B^{s_i}$.

(v) The winner of the game is the winner of the r-round Ehrenfeucht-Fraïssé game (see Definition 2.1.83) played on the structures $(\mathcal{A}, \mathcal{C}_1, ..., \mathcal{C}_p)$ and $(\mathcal{B}, \mathcal{D}_1, ..., \mathcal{D}_p)$.

This game is for existential second-order logic. For universal second-order logic the structures the Duplicator chooses are swapped around, i.e. the Spoiler picks relations on a structure $\mathcal{A} \notin K$ and Duplicator on a structure $\mathcal{B} \in K$. For arbitrary second-order logic the Duplicator picks both structures and the Spoiler then chooses which one to play on.

The key fact about this game is that the Duplicator can choose a structure in step (iii) in response to the relations that the Spoiler has chosen. This game was originally developed and used by Ajtai and Fagin to prove that connectivity is not in monadic ESO even in the presence of some built-in relations [AF90], it was then extended to total orderings by Schwentick in [Sch96] and the proof simplified in [AF97] by removing the random graph

theory part. As mentioned in [Fag96] no results beyond monadic second order logic, and hence so called monadic NP and monadic co-NP [FSV95], are known.

2.2 Optimisation Specific Definitions

A decision problem asks whether a structure exhibits a particular property. If a sentence φ characterises the class of structures K, then $\mathcal{A} \in K \Leftrightarrow \mathcal{A} \models \varphi$, for some structure $\mathcal{A}$ appropriate to φ. In the domain of decision problems, the answers to problems are all binary; $\mathcal{A}$ is either in the class K or not. Optimisation problems extend this binary classification by quantifying how well some structure $\mathcal{A}$ fits into a class; rather than structures being either a member of a class or not, they are now assigned a numeric value corresponding to their optimal solution.

2.2.1 Formal Definition of an Optimisation Problem

Since this thesis is concerned with finite model theory, the rationale for the definition of an optimisation problem naturally comes from Descriptive Complexity arguments. As shall be seen in Chapter 3, the primary aim of the research into optimisation problems is to generate theorems and frameworks for describing optimisation problems using finite model theory; this is also called the descriptive complexity of optimisation problems.

Before going further, the concept of a *solution* to an NP decision problem needs to be introduced.

2.2.1.1 Witnesses and Solutions to NP Decision Problems

Fagin's Theorem (see 2.1.75) is the cornerstone of descriptive complexity and so is always a good place to start when applying it to other domains, in this case optimisation. Recall that due to Fagin's Theorem, an NP decision problem can be written in the form:

$$\mathcal{A} \models \exists \bar{S} \, \forall \bar{x} \, \exists \bar{y} \, \psi(\bar{S}, \bar{x}, \bar{y}) \tag{2.2}$$

where ψ is a quantifier-free first-order formula and the structure $\mathcal{A}$ is the input instance; both are appropriate to and of the same vocabulary τ. Since the outermost (second-order) quantification in Formula 2.2 is existential, only one particular assignment to $\bar{S}$ needs to be found such that the first-order formula $\forall \bar{x} \, \exists \bar{y} \, \psi(\bar{S}, \bar{x}, \bar{y})$ is true. Now assume that ψ has been given and the arities of $\bar{x}$ and $\bar{y}$, along with $\bar{S}$ have been defined. Let σ be the vocabulary consisting solely of the relation symbols from the tuple $\bar{S}$ and let the σ-structure $\mathcal{S}$ be an assignment to the relations such that $(\mathcal{A}, \mathcal{S}) \models \forall \bar{x} \, \exists \bar{y} \, \psi$. $\mathcal{S}$ is said to be a *witness of*, or a *solution to*, the formula 2.2 being true on the structure $\mathcal{A}$.

Only one such witness $\mathcal{S}$ is required to show that a Σ^r_1 formula is true on some structure $\mathcal{A}$, but obviously the selection of one witness does not preclude the existence of others. Note, in order to show that $\mathcal{A} \not\models \exists \bar{S} \, \forall \bar{x} \, \exists \bar{y} \, \psi$ holds (i.e. the structure $\mathcal{A}$ does *not* satisfy the formula) it is necessary to show that there are no witnesses at all.

The concept of a witness is very similar to that of a *short certificate.* To see an example of a short certificate consider the problem Hamiltonian-Path, which given a graph asks the question: *does the graph contain a path that visits every vertex exactly once?* This

problem is in NP and is in fact NP-complete (problem GT39 in Appendix A1.3 of [GJ79]). Now, if given a graph and a path, it can be quickly checked whether or not the path visits every vertex in the graph exactly once; such a path is a solution or a *short certificate* to HAMILTONIAN-PATH for that graph. A witness $\mathcal{S}$ can also be quickly checked for validity against an input structure $\mathcal{A}$, since the formula being checked is first-order as the second-order quantifiers have been removed and replaced by a fixed interpretation of their relation variables. So it can be seen that a witness to an NP decision problem in the finite model theory context is the same as a short certificate in the computational complexity context. For further information on short certificates and their complements, called short disqualifications, see Chapter 9 of [Pap94].

Definition 2.2.1. For an input instance $\mathcal{A}$ and a Σ^r_1-sentence $\exists \bar{S}\, \varphi$, the set of all witnesses $W_\varphi(\mathcal{A})$, is defined as:

$$W_\varphi(\mathcal{A}) := \{\mathcal{S} \mid \text{for each } i, S_i^{\mathcal{S}} \subseteq A^{r_i}, (\mathcal{A}, \mathcal{S}) \models \varphi\}$$

where r_i is the arity of each relation S_i from the tuple $\bar{S}$; the structure $\mathcal{S}$ is of the vocabulary σ which contains only the relation symbols in $\bar{S}$; the formula φ is a first-order formula of the form $\forall \bar{x}\, \exists \bar{y}\, \psi$, with ψ being quantifier-free and $\text{free}(\psi) = \{\bar{x}, \bar{y}, \bar{S}\}$; $\mathcal{A}$ is a structure in the fixed vocabulary τ that is appropriate to the sentence $\exists \bar{S}\, \varphi$.

A decision problem can be rephrased as asserting whether the condition $|W_\varphi(\mathcal{A})| \geq 1$ is true; this condition expresses that there is at least one witness or solution to a sentence of the form $\exists \bar{S}\, \varphi$, which characterises the problem.

On the other hand, optimisation problems are concerned with finding the 'best' solution from the set $W_\varphi(\mathcal{A})$. In order to find the 'best' solution, a numeric cost is calculated using the *objective function*, which maps a tuple $(\mathcal{A}, \mathcal{S})$ consisting of an input instance and a witness/solution, to a numeric cost. The optimal solution is the member of the witness set with the 'best' value (normally either the largest or smallest cost); if $|W_\varphi(\mathcal{A})| = 0$ then there are no solutions to the problem and hence no optimal solution.

This tuple, $(\mathcal{A}, \mathcal{S})$, is also important in evaluating a decision problem, as a sentence $\exists \bar{S}\, \varphi$ characterising a problem is true for some structure $\mathcal{A}$ iff the first-order component φ of the sentence is satisfied by the tuple $(\mathcal{A}, \mathcal{S})$, that is $(\mathcal{A}, \mathcal{S}) \models \varphi$ and thus $\mathcal{S}$ is a witness to the formula $\exists \bar{S}\, \varphi$ being satisfied by the structure $\mathcal{A}$.

2.2.1.2 NP-Optimisation Problems

In general, an optimisation problem will search for the solution with either the largest or smallest cost, giving rise to two distinct types of optimisation, respectively called maximisation and minimisation. Every optimisation problem has a decision variant in which rather than asking for the optimum value, it is asked whether the optimum value is $\geq$ some constant k for a maximisation problem (respectively, $\leq k$ for a minimisation problem). The complexity class that an optimisation problem's decision variant lies in is key in classifying the complexity of the optimisation problem.

Given the discussion of witnesses/solutions above, a formal definition of NP-optimisation problems can now be given.

Definition 2.2.2. An NP-optimisation problem $\mathcal{Q}$ is given by the sets and functions $I_{\mathcal{Q}}$, $F_{\mathcal{Q}}$, $\text{cost}_{\mathcal{Q}}$ and μ, written as the tuple $\mathcal{Q} = (I_{\mathcal{Q}}, F_{\mathcal{Q}}, \text{cost}_{\mathcal{Q}}, \mu)$, where:

(i) $I_{\mathcal{Q}}$ is a class of structures (of some fixed vocabulary τ) acceptable in polynomial time corresponding to the *input instances* of the problem $\mathcal{Q}$.

(ii) $F_{\mathcal{Q}}(\mathcal{A})$ is a function whose domain is $I_{\mathcal{Q}}$ and range is the set of feasible solutions for $\mathcal{A}$; the feasible solutions it maps onto are all recognisable in polynomial time. This function has direct parallels with $W_{\varphi}(\mathcal{A})$, the set of witnesses for a second-order existential formula (see Definition 2.2.1).

(iii) $\text{cost}_{\mathcal{Q}}(\mathcal{A}, S)$ is the *objective function*, which maps a given input structure and solution to a numeric value. It is of the type $\bigcup_{\mathcal{A} \in I_{\mathcal{Q}}} [\{\mathcal{A}\} \times F_{\mathcal{Q}}(\mathcal{A})] \to \mathbb{N}$ and can be computed in polynomial time.

(iv) $\mu \in \{\max, \min\}$

(v) $\text{opt}_{\mathcal{Q}}(\mathcal{A})$ is a function that returns the optimal value of the problem $\mathcal{A}$ for some input structure $\mathcal{A} \in I_{\mathcal{Q}}$. It is defined as

$$\text{opt}_{\mathcal{Q}}(\mathcal{A}) := \mu\{\text{cost}_{\mathcal{Q}}(\mathcal{A}, S) \mid S \in F_{\mathcal{Q}}(\mathcal{A})\}$$

(vi) $\mathcal{Q}_D$ is the decision problem variant of $\mathcal{Q}$: given some structure $\mathcal{A} \in I_{\mathcal{Q}}$ and a constant $k \in \mathbb{N}$, is there a solution $S \in F_{\mathcal{Q}}$ s.t. if $\mu = \max$ then $\text{cost}_{\mathcal{Q}}(\mathcal{A}, S) \geq k$, or if $\mu = \min$ then $\text{cost}_{\mathcal{Q}}(\mathcal{A}, S) \leq k$? The decision problem $\mathcal{Q}_D$ is in NP.

The class of all such optimisation problems is called NP_{opt}. It takes its name from the observation that the decision variant of any problem in the class NP_{opt} is in NP. It should be noted that every optimisation problem has a decision variant (as per point (vi) above), but some decision problems have no optimisation variant. For example the question "is x prime?" has no optimisation variant, as x is either prime or not – there is no concept in number theory as to how good a prime x is. This leads to the relationship $\alpha(\mathsf{NP}_{opt}) \subset \mathsf{NP}$, where α is a function that transforms a set of optimisation problems into a set of their corresponding decision variants.

Remark 2.2.3. Note that according to Definition 2.2.2, *every* optimisation problem is in fact an NP-optimisation problem, irrespective of condition (vi). This is because the problem of computing the optimal value, as specified in condition (v) of Definition 2.2.2, is in fact an NP-optimisation problem. To see this, observe the following algorithm for the decision variant of an optimisation problem: guess a feasible solution S and then check whether it is greater or equal to than (for maximisation) or less than or equal to (for minimisation) the target value k. Clearly, as all feasible solutions are recognisable in polynomial time and the cost function can be calculated in polynomial time, this problem is in NP, hence rendering condition (vi) redundant. However, it is included as it appears in analogous definitions in [KT94, KT95]; for in these definitions the objective function is defined to be computable in time polynomial in the size of the input, i.e., the instance and a feasible solution, rather than in the size of the instance.

Remark 2.2.4. Definition 2.2.2 implies that all feasible solutions to some instance can be taken to have size bounded by some polynomial in the size of the instance. Given that the notion here of a solution to an optimisation problem is simply that a numeric value should be found and not a witnessing feasible solution, there is no real need to discuss the computational nature of a set of feasible solutions corresponding to some instance. In particular, Definition 2.2.2 says nothing about the complexity of deciding whether some potential feasible solution is indeed an actual feasible solution to some instance. It turns out that most (instances of) natural optimisation problems have easily recognisable sets of feasible solutions.

2.2.1.3 Absence of any Feasible Solutions

In some situations there are no feasible solutions to an optimisation problem for some given input structure. To see this in the context of a real example consider the problem SHORTEST-PATH, which given a graph and two named vertices s and t, asks the question: what is the shortest path between s and t? (See Definition 2.2.3.5 for a more formal description). A feasible solution to this problem is a path from s to t; if there is no path between s and t in the graph then there are no feasible solutions. This means that the set over which the minimum is taken (since Shortest Path is a minimization problem) in point (v) of Definition 2.2.2 is empty. To deal with this situation the following is needed:

Definition 2.2.5. The maximum and minimum of the empty set is undefined, represented by the symbol $\perp$. More specifically:

- $\max\{\} \equiv \max \emptyset \equiv \perp$
- $\min\{\} \equiv \min \emptyset \equiv \perp$

So, if the set $F_{\mathcal{Q}}(\mathcal{A})$ is empty then the optimal solution is undefined, written $\text{opt}_{\mathcal{Q}}(\mathcal{A}) := \perp$.

Remark 2.2.6. When the optimal solution of problem $\mathcal{Q}$ is undefined on input structure $\mathcal{A}$, it is the case that for the decision variant $\mathcal{Q}_D$ of the problem the structure $\mathcal{A}$ is a *no instance*, irrespective of the value chosen for k (see point (vi) of Definition 2.2.2). This is because there is no such feasible solution $S \in F_{\mathcal{Q}}$ that satisfies the predicate $\text{cost}_{\mathcal{Q}}(\mathcal{A}, S) \geq k$ (or $\leq k$ if $\mathcal{Q}$ is a minimisation problem), simply because there are no feasible solutions at all.

2.2.1.4 Classes of NP-optimisation Problems

With the class of all NP-optimisation problems dealt with in Definition 2.2.2, it is now prudent to introduce the other main classes of optimisation problems.

Definition 2.2.7. The class of NP-*optimisation problems* NP_{opt} is composed from two distinct sub-classes, NP-*maximisation problems* NP_{max}, and NP-*minimisation problems* NP_{min}, which respectively correspond to the cases when $\mu = \max$ and $\mu = \min$ in the tuple that defines an NP-optimisation problem (see Definition 2.2.2).

Up until now the value of the cost of a solution hasn't been mentioned, other than the fact that it is a natural number. Whilst arbitrary natural numbers are usually encoded on

a computer using a binary string whose length is the base 2 logarithm of the number, it is not clear exactly how such numbers are encoded using relational structures. A straight forward way of doing this is to measure the cardinality of a relation but this would only allow numbers whose size is bounded by a polynomial in the size of input to be represented. It turns out (as will hopefully become clear later in Chapter 3) that this is a reasonable way to measure the cost of a feasible solution in many natural optimisation problems and so the following sub-class of problems is a useful restriction:

Definition 2.2.8. An optimisation problem $\mathcal{Q}$ is *polynomially-bounded* if there is a polynomial q such that for every instance $\mathcal{A}$ in $I_{\mathcal{Q}}$, $\text{opt}(\mathcal{A}) \leq q(|\mathcal{A}|)$ (note that in general the value $\text{opt}(\mathcal{A})$ of some instance $\mathcal{A}$ might be exponential in $|\mathcal{A}|$). The (sub-)class of polynomially-bounded NP-optimisation problems is denoted by NP_{opt}^{PB}, the (sub-)class of polynomially-bounded NP-maximisation problems by NP_{max}^{PB}, and the (sub-)class of polynomially-bounded NP-minimisation problems by NP_{min}^{PB}.

As shall be seen in Chapter 3 the polynomially-bounded restriction can be lifted by adding to the logical framework a function that encodes numbers whose value is exponential in the size of the input; such an encoding is technically awkward to use and as such is left out of the initial theorems (which are thus restricted to polynomially-bounded optimization problems).

2.2.1.5 Complete Problems

Proposition 2.2.9. *If the decision variant $\mathcal{Q}_D$ of an* NP*-optimisation problem $\mathcal{Q}$ is* NP*-complete, then $\mathcal{Q}$ is* NP*-hard.*

This is direct from the definition of an NP-hard problem (see Definition 2.1.53): use $\mathcal{Q}_D$ as an oracle and apply a binary search algorithm to find the optimal value to $\mathcal{Q}$. The binary search requires a number of queries to the oracle that is logarithmic in the size of the largest possible cost, which in the general (non-polynomially-bounded) case is exponential in the size of the input. This means it makes at most a polynomial number of calls to the oracle and so is NP-hard.

A problem is complete for a class if it is amongst the hardest problems in that class; if an optimisation problem is NP-hard then this means that there is polynomial-time Turing reduction from some NP-complete problem to the optimisation problem. Since all NP-optimisation problems must have a decision problem in NP, the hardest problems are those with a decision problem that is NP-complete.

Definition 2.2.10. An optimisation problem $\mathcal{Q}$ is NP_{opt}-complete iff it is NP-hard.

There are polynomial-time reductions between all NP-complete problems and so it immediately follows that there are polynomial-time Turing reductions between all NP_{opt}-complete problems.

2.2.2 P-optimisation Problems

The definition of the class of P-optimisation problems (usually abbreviated to P_{opt}) is identical to that of NP-optimisation problems except for one restriction regarding the class that the decision variant lies in:

Definition 2.2.11. A P-optimisation problem $\mathcal{Q}$ is given by the sets and functions $I_{\mathcal{Q}}$, $F_{\mathcal{Q}}$, $\text{cost}_{\mathcal{Q}}$ and μ, written as $\mathcal{Q} = (I_{\mathcal{Q}}, F_{\mathcal{Q}}, \text{cost}_{\mathcal{Q}}, \mu)$, where points (i) to (v) of Definition 2.2.2 hold and with point (vi) is given as:

(vi) $\mathcal{Q}_D$ is the decision problem variant of $\mathcal{Q}$: given some structure $\mathcal{A} \in I_{\mathcal{Q}}$ and a constant $k \in \mathbb{N}$, is there a solution $\bar{S} \in F_{\mathcal{Q}}$ s.t. if $\mu = \max$ then $\text{cost}_{\mathcal{Q}}(\mathcal{A}, \bar{S}) \geq k$, or if $\mu = \min$ then $\text{cost}_{\mathcal{Q}}(\mathcal{A}, \bar{S}) \leq k$. The decision problem $\mathcal{Q}_D$ is in P.

The class of all such problems is called P_{opt} and since $\mathsf{P} \subseteq \mathsf{NP}$ then $\mathsf{P}_{opt} \subseteq \mathsf{NP}_{opt}$. The restriction of the class P_{opt} to those with *polynomially bounded* objective functions (see Definition 2.2.8) is called P^{PB}_{opt}, where naturally $\mathsf{P}^{PB}_{opt} \subset \mathsf{P}_{opt}$. All the classes exhibit a dichotomy between *maximisation* and *minimisation* problems; these are denoted (in a manner similar to that for NP-optimisation problems) as P_{max}, P_{min}, P^{PB}_{max} and P^{PB}_{max}.

Importantly, the *solution* of an optimization problem $\mathcal{Q} = (I_{\mathcal{Q}}, F_{\mathcal{Q}}, \text{cost}_{\mathcal{Q}}, \mu)$ is an algorithm that given any instance $\mathcal{A}$ of the problem, produces as output the value $\text{opt}(\mathcal{A})$ and *not* (necessarily) an optimal feasible solution from $F_{\mathcal{Q}}(\mathcal{A})$ (if there is one). In fact, this algorithm need not even work with representations of feasible solutions; all it has to do is to come up with the optimal value. Note that all problems $\mathcal{Q} = (I, F, \text{cost}, \mu)$ in P_{opt} can be solved in polynomial-time, for: given any instance $\mathcal{A}$ of size $|A|$ and any feasible solution $\mathcal{S} \in F(\mathcal{A})$, by definition $\text{cost}(\mathcal{A}, \mathcal{S})$ is $O(2^{p(|A|)})$, where p is some polynomial. Now there is according to condition (vi) of Definition 2.2.11 an algorithm that runs in time $q(|A|)$, where q is some polynomial, that decides whether the optimal solution is greater than (or less than) an input parameter k. Now, by applying this decision algorithm in conjunction with a binary search (see e.g. [Knu73, CLRS09]) the optimal cost can be found in $O(\log_2(2^{p(|A|)}).q(|A|)) = O(p(|A|).q(|A|))$, hence this combination yields a polynomial-time algorithm for computing $\text{opt}(\mathcal{A})$.

In terms of existing complexity classes, P_{opt} is the same as the subset of problems in FP (see Chapter 10 of [Pap94]) where the output structure is a positive natural number.

2.2.3 Some Common Optimisation Problems

When discussing logical frameworks for describing problems it is useful to give examples of their application to concrete problems. Here are the definitions of some common optimisation problems that shall be used for this purpose.

2.2.3.1 Max-Flow, Max-Flow^{PB} and Max-Unit-Flow

Consider the maximum flow problem Max-Flow $= (I, F, \text{cost}, \max)$, where:

- I is the set of tuples $(\mathcal{G}, w, s, t)$, with $\mathcal{G}$ a digraph and s and t two named vertices of $\mathcal{G}$, where s has in-degree 0 and t has out-degree 0. The relation w maps each edge in $\mathcal{G}$ to a weight, effectively making $\mathcal{G}$ a *weighted* digraph. An example of how such a weight relation may be encoded is discussed in Section 3.7.1.1.

- $F : (G, w, s, t) \mapsto \{P_i\}$ is a relation from input instances to valid flows, where a flow P is an assignment of values to each edge in the digraph $\mathcal{G}$ under the constraints that no value exceeds the weight assigned to the edge by w and that the sum of the

values incident upon a vertex is equal to the sum of the values radiating out of it. $F((\mathcal{G}, w, s, t))$ is the set of all valid flows for $(\mathcal{G}, w, s, t)$.

- $\text{cost}(\mathcal{A}, \mathcal{S})$, for some instance $\mathcal{A} \in I$ and for some feasible solution $\mathcal{S} \in F(\mathcal{A})$, is the size of the flow $\mathcal{S}$, that is the sum of the flows on the edges emanating out of s (which is equal to the sum of the flows on the edges incident on t in any valid flow).

It is well-known that the decision version of MAX-FLOW is in P (see, e.g. [CLRS09] or Chapter 1 of [Pap94]); thus, MAX-FLOW $\in \mathsf{P}_{max}$ but note that MAX-FLOW is, in general, not in P_{max}^{PB} since the weights on the edges can be at most exponential in the size of the input (as it is possible to encode such numbers of the order of $O(2^n)$ with a sequence of symbols of length n). If the weights are explicitly restricted to being bounded by some polynomial in the size of the input then the problem would be in fact be in P_{max}^{PB}; call the polynomially bounded restriction of the problem MAX-FLOW^{PB}. Such a restriction can be achieved by considering the maximum flow problem MAX-UNIT-FLOW $= (I_U, F_U, \text{cost}_U, \max)$, which is a restriction of MAX-FLOW to only have unit weights on the edges; effectively removing the weight mapping w. Or more formally:

- I_U is the set of tuples $(\mathcal{G}, s, t)$, with $\mathcal{G}$ a digraph and s and t two named vertices of $\mathcal{G}$, where s has in-degree 0 and t has out-degree 0. $\mathcal{G}$ is an *unweighted* digraph.

- $F_U : (\mathcal{G}, s, t) \mapsto \{P_i\}$ is a relation from input instances $(\mathcal{G}, s, t)$ to sets of all possible flows $\{P_0, ..., P_m\}$ for each instance (where m depends on the instance).

with the cost function being the same as in MAX-FLOW.

The problem MAX-UNIT-FLOW $\in \mathsf{P}_{max}^{PB}$ since the maximum possible value of a flow is equal to the number of edges emanating from the source vertex s, which is a quantity that is linearly bounded by the number of vertices in the graph. Also note that a MAX-FLOW instance $\mathcal{A}$ that has the values of its weight mapping $w_{\mathcal{A}}$ polynomially bounded can be transformed into an instance $\mathcal{B}$ of the problem MAX-UNIT-FLOW by simply replacing each weighted edge with a sequence of edges whose length is equal to the weight of the original edge. This transformation introduces more vertices to the graph, the number of which is bounded by the same polynomial that bounds the weight mapping $w_{\mathcal{A}}$. Since it takes a polynomial amount of time to construct the MAX-UNIT-FLOW instance from the polynomially-bounded MAX-FLOW instance the computational complexity of the problem (transformation plus solving) is unchanged, whereas if the weight mapping was *exponentially bounded* then the transformation would take an exponential amount of time to run, changing the computational complexity of the problem.

Using the notation introduced in Section 2.1.2.2 this transformation (c.f. reduction) means that MAX-UNIT-FLOW $\leq$ MAX-FLOW^{PB}.

2.2.3.2 MAX-2-SAT

Firstly, recall the decision problem SAT (whose name is a shortening of *satisfiability)* from Section 2.1.3. Famously this is the first ever NP-complete problem [Coo71], which takes as its input a boolean formula in conjunctive normal form over a set of *literals* (boolean variables that can be assigned either true or false) and asks: is there an assignment of

truth values to the literals such that all the clauses in the formal are true? A commonly used restriction of SAT is 3-SAT, in which each clause has exactly 3 literals; this decision problem is also NP-complete [Coo71]. Another restriction is 2-SAT, in which each clause has exactly 2 literals; this decision problem is in P.

Moving away from decision problems and back to optimisation, the most common optimisation version of satisfiability problems is MAX-SAT (see Definition 2.1.70), which asks for the maximum number of clauses that can be satisfied in a formula.

Consider the maximum 2-satisfiability problem MAX-2-SAT $= (I, F, \text{cost}, \max)$, where:

- I is the set of conjunctive normal form formulae φ where every clause has 2 literals;
- $F(\varphi)$ is the set of truth assignments on the Boolean variables involved in φ;
- $\text{cost}(\varphi, \mathcal{S})$, for some instance $\varphi \in I$ and for some feasible solution $\mathcal{S} \in F(\varphi)$, is the number of clauses made true in φ under the truth assignment $\mathcal{S}$.

The decision version of this problem, k-MAX-2-SAT is NP-complete [GJS76], unless $k = |C|$, where C is the set of clauses in the formula φ in the input instance, because then the problem is effectively the same as the decision problem 2-SAT. Since k-MAX-2-SAT is in NP it follows by Definition 2.2.2 that MAX-2-SAT is in NP_{max} but not P_{max} unless $\mathsf{P} = \mathsf{NP}$, as by Definition 2.2.11 MAX-2-SAT is in P_{max} iff k-MAX-2-SAT is in P (which would require that $\mathsf{P} = \mathsf{NP}$ since k-MAX-2-SAT is NP-complete).

Observe that the maximum number of clauses in a formula φ in I is equal to half the length of the formula, and so the values of the cost function are polynomially bounded, in fact $\text{cost}(\varphi, \mathcal{S}) \leq \frac{|\varphi|}{2} \leq c.|\varphi|$, where $|\varphi|$ is the length of the formula φ; c is some rational constant; $\varphi \in I$ and $\mathcal{S} \in F(\varphi)$. This shows that the values returned by the cost function are linear in the length of the input formula and hence linear in the size of the input. Hence MAX-2-SAT is in NP^{PB}_{max}.

2.2.3.3 MAX-2-HORN-SAT

The problem MAX-2-HORN-SAT is obtained by taking the problem MAX-2-SAT and adding the restriction that all instances must be Horn formulae (see Section 2.1.3.1). Due to a result in [JS87], where the decision variant of MAX-2-HORN-SAT, k-MAX-2-HORN-SAT is shown to be NP-complete, it follows that even with the Horn restriction MAX-HORN-2-SAT is in NP^{PB}_{max}.

2.2.3.4 MAX-2-SAT($\leq$2)

Define the problem MAX-2-SAT($\leq$2) by restricting instances of MAX-2-SAT so that every variable appears in at most 2 clauses. The decision version k-MAX2SAT($\leq$2) can be solved in linear time [RRR98]. Therefore, since k-MAX-2-SAT($\leq$2) is in P, it follows by Definition 2.2.11 that the optimisation problem MAX-2-SAT($\leq$2) is in P^{PB}_{max}.

2.2.3.5 SHORTEST PATH

Shortest path is part of a family of problems that operate on graphs (and digraphs) $\mathcal{G}$ that contain two named vertices: s (the *source* or *start* vertex) and t (the *sink* or *end*

vertex). Once such decision problem in the family is REACHABILITY, which asks: is there a path between vertices s to t in $\mathcal{G}$? This can be extended with a constant k to ask: is there a path between vertices s and t in G such whose length is $\leq k$? Call this decision problem the k-minimum shortest path problem or k-MIN-SP for short. Now consider the optimisation version MIN-SP $= (I, F, \text{cost}, \min)$, where:

- I is the set of triples $(\mathcal{G}, s, t)$, with $\mathcal{G}$ a digraph and s and t two named vertices of $\mathcal{G}$;
- $F : (\mathcal{G}, s, t) \mapsto \{P_i\}$ is a relation mapping input instances $(\mathcal{G}, s, t)$ to the set of all possible paths $\{P_0, ..., P_m\}$ in $\mathcal{G}$ from s to t (where m depends on the input instance).
- $\text{cost}(\mathcal{A}, \mathcal{S})$, for some instance $\mathcal{A} \in I$ and for some feasible solution $\mathcal{S} \in F(\mathcal{A})$, is defined as the length of the path $\mathcal{S}$.

It is well-known that the decision version of MIN-SP is in P (see, e.g. [CLRS09]); thus, since the length of the longest possible path in $\mathcal{G}$ is $|G| - 1$ (where $|G|$ is the number of vertices in $\mathcal{G}$) it follows by Definition 2.2.11 that MIN-SP $\in \mathsf{P}^{\,PB}_{min}$.

2.2.4 Extracting Parameters from Fixed-Point Operations

This section deals with metrics associated with fixed-point operators.

2.2.4.1 Inductive depth operator for LFP and IFP

As mentioned earlier in Section 2.1.1.6, the *inductive depth* of an operator (for some structure $\mathcal{A}$), is the number of iterations required before a fixed point is encountered. As will be discussed later, the inductive depth of an operator is an important measure and as such, the following notation is introduced:

$$\mathit{depth}\left([\mathrm{LFP}_{R,\bar{x}}\,\varphi]\,(\bar{t})\right) \in \mathbb{N} \cup \{\bot\}$$

where if $\mathcal{A} \models [\mathrm{LFP}_{R,\bar{x}}\,\varphi](\bar{t})$ then the above is evaluated as the inductive depth of the least fixed point operator applied to the structure $\mathcal{A}$, *minus one*. Conversely, if $\mathcal{A} \not\models [\mathrm{LFP}_{R,\bar{x}}\,\varphi](\bar{t})$ then the above is evaluated as $\bot$, which is read as being undefined (even though the inductive depth of the LFP operator is).

An inductive depth of zero means that the least fixed point R is $\emptyset$, implying that $\mathcal{A} \not\models [\mathrm{LFP}_{R,\bar{x}}\,\varphi](\bar{t})$ since $\bar{t} \notin \emptyset$. By subtracting one from the inductive depth of an operator the values of the depth will be in the range 0 to $|A|^k - 1$ (or $\bot$).

As well as knowing the value of the relation R at the least fixed point it is also useful to be able to refer to its value at a particular point in the sequence before the least fixed point is reached. For this the following notation is used:

$$X^i = \left[\mathrm{LFP}^i_{R,\bar{x}}\,\varphi\right]$$

which refers to the i^{th} relation constructed after i applications of the least fixed point operator generated by the formula φ. Let the inductive depth of the operator defined by φ applied to some structure $\mathcal{A}$ be r. If $i > r$ then $X^i = X^r$. Since this holds true for all integers larger than the inductive depth, it is written that $X^\infty = X^r$.

Both the inductive depth and i^{th} iteration notations are applied to IFP operators in exactly the same way that they are to LFP operators.

2.3 (Hybrid) Modal Logic Specific Definitions

In this section are the preliminary definitions used in Chapter 4. They start with an introduction to modal logic, specifically basic modal logic (2.3.1) and develop this into hybrid modal logic, before fixing on the logic of interest: Hybrid Graph Logic (2.3.2). Finally, the concept of bisimulation is then introduced (2.3.3).

The definitions given here intend to build upon those given in Section 2.1 regarding relational structures (2.1.2), descriptive complexity (2.1.4), logical equivalence classes (2.3.2.5) and games (2.1.5).

2.3.1 Basic Modal Logic

Modal logic differs from first-order logic (see Definition 2.1.7) in that the models/structures that it operates on are far less general. Classically, modal logic is the extension of propositional logic with two unary operators $\Box$ and $\Diamond$ that are used to express that a property expressed by a formula φ is respectively necessarily or possibly true. In modern descriptions of modal logic (see e.g. [HC96, BdRV01, GO07]) the underlying structure is a graph and these operators traverse it examining the truth of propositions at each vertex. The structures of modal logic are formally defined as a special case of the more general τ-structure, as introduced in Definition 2.1.2.

Definition 2.3.1. In modal logic, a *model* $\mathcal{M}$ is a relational structure with universe W and vocabulary $\langle R^2, \mathsf{P}\rangle$, where $\mathsf{P} = \langle P_1^1, P_2^1, ..., P_l^1\rangle$ is the set of l *proposition symbols*, or simply the set of *propositions*.

In essence, a model in modal logic is a directed graph (W, R) with a set of colours P applied to the vertices. The uncoloured graph (W, R) is called a *modal frame* or sometimes just a *frame.* As a model is parameterised by the set of propositions P it is sometimes referred to as a P-structure (similar in naming to that of τ-structures from Definition 2.1.2).

Definition 2.3.2. Formulae in *basic modal logic* are inductively defined in a similar way to those of first-order logic (see Definition 2.1.7). The well-formed atomic formulae of modal logic are p, where p is a proposition symbol and $\bot$. If ψ and ϕ are well-formed formulae then so are $\psi \Rightarrow \phi$ and $\Diamond\psi$. The class of logic ML(P) consists of all such basic modal logic formulae where the proposition symbols are contained within the set P.

A basic modal logic formula is evaluated on a model at a particular point or vertex (this is what makes the logic *modal*) in the underlying modal frame. The key part of the evaluation is to examine whether a proposition p holds for some vertex v in $\mathcal{M}$. For this reason the concept of the *valuation function* $\mu : \mathsf{P} \to \mathcal{P}(W)$ is introduced, which maps a proposition symbol $p \in \mathsf{P}$ to a set of vertices $\mu(p)$. It should be noted that in a model $\mathcal{M}$ the set of vertices returned by the valuation function $\mu(p_i)$ is equal to the instantiation of the unary relation symbol p_i in $\mathcal{M}$ to $p_i^{\mathcal{M}}$. As will be explained later, when dealing with

frame satisfaction this separation of the frame from the values of the proposition symbols in the model is important.

The modal frame of a model is in fact a directed graph and so the universe M of a model $\mathcal{M}$ is the set of vertices in the graph (or the unary relation W from Definition 2.3.1) and a modal frame $\mathcal{G}$ is a directed graph structure of vocabulary τ_g (see Definition 2.1.5). With this definition of graphs and the concept of the valuation function it can be seen that a modal P-structure $\mathcal{M}$ is a tuple of a graph structure and an valuation function appropriate to P, i.e. $\mathcal{M} = \langle \mathcal{G}, \mu \rangle$.

Definition 2.3.3. A *pointed modal structure* or a *pointed model* is simply a structure $\mathcal{M} = \langle \mathcal{G}, \mu \rangle$ combined with a point (or vertex) from the underlying modal frame $v \in M$. This is commonly denoted $(\mathcal{M}, v)$ (see structure extension in Definition 2.1.1.1) but can just as well be denoted $(\langle \mathcal{G}, \mu \rangle, v)$, $\langle \langle \mathcal{G}, \mu \rangle, v \rangle$ or simply just $(\mathcal{G}, \mu, v)$, following the convention of using lower-case Greek letters for functions and lower-case Roman letters for constants in order to avoid ambiguity. A *pointed (modal) frame* is a modal frame $\mathcal{G}$ combined with some point (or vertex) $v \in G$, denoted $(\mathcal{G}, v)$.

The terms 'digraph' and 'frame' and the terms 'vertex' and 'point' shall be used interchangeably, depending upon the context.

Remark 2.3.4. The universe of a model $\mathcal{M} = \langle \mathcal{G}, \mu \rangle$ is equal to the universe of the underlying frame (i.e. $M = G$). Since G is the set of vertices of a graph it is usually given the symbol V; when a set V is referred to, it is the universe of the model.

The semantics for evaluating whether a model $\mathcal{M} = \langle \mathcal{G}, \mu \rangle$ is satisfied by a formula $\varphi \in \mathrm{ML}(\mathsf{P})$ are:

Definition 2.3.5. Let φ be a formula from the modal logic $\mathrm{ML}(\mathsf{P})$, where P is a set of proposition symbols, and let $\mathcal{M} = \langle \mathcal{G}, \mu \rangle$ be a model appropriate to the modal system $\mathrm{ML}(\mathsf{P})$. The satisfaction relation $\models$ is defined as follows:

(i) $(\mathcal{M}, v) \models p$, where $p \in \mathsf{P}$, iff $v \in \mu(\mathsf{P})$.

(ii) It is not the case that $(\mathcal{M}, v) \models \bot$ is true.

(iii) If φ is of the form $\psi \Rightarrow \psi'$ then $(\mathcal{M}, v) \models \varphi$ if, and only if, whenever $(\mathcal{M}, v) \models \psi$, it must be he case that $(\mathcal{M}, v) \models \psi'$.

(iv) If φ is of the form $\Diamond\psi$ then $(\mathcal{M}, v) \models \varphi$ if, and only if, there exists some point v' of $\mathcal{G}$ for which $E(v, v')$ holds in $\mathcal{G}$ and for which $(\mathcal{M}, v') \models \psi$.

If $(\mathcal{M}, v) \models \varphi$ then it is said that $\mathcal{M}$ *satisfies* φ at v. If it is not the case that $(\mathcal{M}, v) \models \varphi$, written $(\mathcal{M}, v) \not\models \varphi$, then it is said that $\mathcal{M}$ is *unsatisfied* by φ at v.

The usual Boolean connectives $\vee, \wedge, \neg$ and $\Leftrightarrow$ can be defined in basic modal logic in the same way that they are defined in first-order logic (see Definition 2.1.7). The short-hand $\top$ is used for $\neg\bot$ and is referred to as *true*. The short-hand $\Box\varphi \equiv \neg\Diamond\neg\varphi$ is also used and is referred to as *box* or the *universal* access operator (with $\Diamond$ being *diamond* or the *existential* access operator).

2.3.1.1 Satisfiability and Validity

A formula in basic modal logic is satisfiable if from some class of P-structures there is a model that satisfies the given formula at a certain point:

Definition 2.3.6. Given a formula $\varphi \in \mathrm{ML}(\mathsf{P})$, where P is a set of proposition symbols, φ is P-*satisfiable* (or just *satisfiable* if P is clear form the context) if, and only if, there exists some model $\mathcal{M}$ appropriate to P and there exists some point v in $\mathcal{M}$ such that $(\mathcal{M}, v) \models \varphi$.

The dual property of validity is similarly defined:

Definition 2.3.7. Given a formula $\varphi \in \mathrm{ML}(\mathsf{P})$, where P is a set of proposition symbols, φ is P-*valid* (or just *valid* if P is clear from the context) if, and only if, for all models $\mathcal{M}$ appropriate to P and for all points v in $\mathcal{M}$ it holds that $(\mathcal{M}, v) \models \varphi$.

Observe that if a formula φ is satisfiable then its negation $\neg\varphi$ is *not valid* (or *invalid*), since the model $\mathcal{M}$ and point v that witness satisfiability of φ also witness the invalidity of $\neg\varphi$. The dual of this relationship is that if a formula φ is valid then its negation $\neg\varphi$ is *not satisfiable* (or *unsatisfiable*). These observations give rise to the following Lemma:

Lemma 2.3.8. *Given an algorithm for deciding* P*-satisfiability and a formula* $\varphi \in \mathrm{ML}(\mathsf{P})$*, the validity problem for* φ *can be answered by negating the algorithm's response for satisfiability of* $\neg\varphi$*: if* $\neg\varphi$ *is satisfiable then* φ *is invalid; if* $\neg\varphi$ *is unsatisfiable then* φ *is valid.*

The above lemma means that the satisfiability problem is decidable (i.e. there is an algorithm that solves it) if, and only if, the validity problem is decidable, since there is a reduction each way between the two problems.

Theorem 2.3.9. *The satisfiability problem for basic modal logic is decidable.*

The direct consequence of the above theorem and lemma is:

Corollary 2.3.10. *The validity problem for basic modal logic is decidable.*

These decidability results come from the fact that basic model logic exhibits the *finite tree model* property; more details can be found in [HC96, Var96, BdRV01, MV07].

2.3.2 Hybrid Modal Logic and Graph Logic

In this section, the syntax and semantics of Hybrid Graph Logic (as it was defined in [BS09]) are recapitulated. In essence, Hybrid Graph Logic is a hybrid modal logic augmented with a facility to validate paths in structures, and the structures in which the formulae of Hybrid Graph Logic are interpreted are modal frames with a single modality. A hybrid modal logic is an extension of modal logic with the concept of *nominals*, which behave like propositional symbols except that they can only refer to exactly one vertex/point in the modal frame. Further information on hybrid modal logics can be found in [AtC07].

2.3.2.1 Syntax

First, the syntax of Hybrid Graph Logic is presented. Since every basic modal logic formula is also a Hybrid Graph Logic formula the definitions of syntax and semantics shall extend those presented in Section 2.3.1. Every formula of Hybrid Graph Logic is parameterised by a set of *proposition symbols* P (as in basic modal logic) and by a set of *nominals* N.

Definition 2.3.11. Given a set P of propositional symbols and a set N of nominals, the formulae of the logic HGL(P, N) (called Hybrid Graph Logic) are inductively defined as those of ML(P) plus the following:

$$n \ ; \ \Diamond^+\psi \ ; \ \text{ and } @_n\psi,$$

where $n \in \mathsf{N}$ is a nominal and $\psi \in \text{HGL}(\mathsf{P}, \mathsf{N})$.

The structures in which formulae of HGL(P, N) are interpreted are the same as those for basic modal logic, except that the valuation function also has evaluations for the nominals.

Definition 2.3.12. Given a set of propositional symbols P and a set of nominals N a $\mathsf{P} \cup \mathsf{N}$-structure (or model) $\mathcal{M} = (\mathcal{G}, \mu)$ in Hybrid Graph Logic comprised of a graph $\mathcal{G}$ and a valuation function $\mu : \mathsf{P} \cup \mathsf{N} \to \mathcal{P}(V)$. The valuation function in HGL is the same as in basic modal logic except that the nominals are always evaluated to a singleton set. If $\mu(n) = \{v\}$, where $n \in \mathsf{N}$ and $v \in V$, then sometimes this is written as $\mu(n) = v$ and the nominal n is said to sit on the point v.

2.3.2.2 Semantics

The semantics of the Hybrid Graph Logic HGL(P, N) are built on top of those of basic modal logic (see Definition 2.3.5), with interpretations of the Hybrid Graph Logic specific syntax given below.

Definition 2.3.13. Let φ be a formula of Hybrid Graph Logic that involves proposition symbols from the set P and nominals from the set N. φ is interpreted in a $\mathsf{P} \cup \mathsf{N}$-structure $\mathcal{M} = \langle \mathcal{G}, \mu \rangle$ as per the semantics of basic modal logic with the following additional rules:

(i) $(\mathcal{M}, v) \models n$ if, and only if, $v = \mu(n)$.

(ii) If φ is of the form $\Diamond^+\psi$ then $(\mathcal{M}, v) \models \varphi$ if, and only if, there exist points $v_0, v_1, \ldots, v_q$, where $q \geq 1$, for which: $v = v_0$; $E(v_i, v_{i+1})$ holds in $\mathcal{G}$, for all $i = 0, 1, \ldots, q-1$; and $(\mathcal{M}, v_q) \models \psi$.

(iii) If φ is of the form $@_n\psi$ then $(\mathcal{M}, v) \models \varphi$ if, and only if, $(\mathcal{M}, u) \models \psi$, where $u = \mu(n)$.

The short-hand $\Box^+\psi$ is used for $\neg\Diamond^+\neg\psi$ (making $\Box^+$ the *dual* of the operator $\Diamond^+$). While the $\Diamond$ operator is used to ask if a formula is true at some vertex that is adjacent to the current evaluation vertex v, the $\Diamond^+$ operator asks if a formula is true at some vertex that is reachable by a path of at least one edge from v. So, $(\mathcal{M}, v) \models \Diamond^+\psi$ is the same as asking if $(\mathcal{M}, u) \models \psi$ holds at some vertex u and that $E^+(v, u)$ is true, where E^+ is the non-reflexive, transitive closure of E.

2.3.2.3 Characterising Problems Using HGL

Hybrid Graph Logic can be used to define digraph problems; that is, classes of digraphs that are closed under isomorphisms. To do this, the concept of *global satisfaction* and *frame validity* are required.

Definition 2.3.14. Let φ be some formula of Hybrid Graph Logic that involves propositional symbols from the set P and nominals from the set N, and let $\mathcal{M} = \langle \mathcal{G}, \mu \rangle$ be some model with modal frame $\mathcal{G}$ and valuation function $\mu : \mathsf{P} \cup \mathsf{N} \to \mathcal{P}(V)$. It is said that the model $\mathcal{M}$ *globally satisfies* φ, written $\mathcal{M} \models \varphi$ if, and only if, at every point v of the modal frame $\mathcal{G}$ it is the case that $(\mathcal{M}, v) \models \varphi$.

Definition 2.3.15. Let φ be some formula of Hybrid Graph Logic from the set $\mathrm{HGL}(\mathsf{P}, \mathsf{N})$ and let $\mathcal{G}$ be a modal frame. The formula φ is *valid* in $\mathcal{G}$ (or φ is $\mathcal{G}$*-valid*), written $\mathcal{G} \models \varphi$, if, and only if, for every possible valuation function $\mu : \mathsf{P} \cup \mathsf{N} \to \mathcal{P}(V)$, it is the case that $\langle \mathcal{G}, \mu \rangle \models \varphi$ (that is, the model composed of the frame $\mathcal{G}$ and the valuation function μ globally satisfies φ).

This idea of a formula being valid on a modal frame is used to define problems in Hybrid Graph Logic in a manner similar to the way that problems are defined using first-order logic (see Definition 2.1.4.1).

Definition 2.3.16. Let φ be some formula of Hybrid Graph Logic. The *problem* defined by φ consists of those frames $\mathcal{G}$ for which φ is valid in $\mathcal{G}$; that is, for which $\mathcal{G} \models \varphi$.

So, a formula φ characterises a graph problem $\mathcal{Q}$ if, and only if, for every graph $\mathcal{G}$, the formula φ is valid in $\mathcal{G}$ when $\mathcal{G}$ corresponds to a yes-instance of $\mathcal{Q}$ and φ is not valid in $\mathcal{G}$ when $\mathcal{G}$ is a no-instance of $\mathcal{Q}$.

Care should be taken when working with formulae of Hybrid Graph Logic in relation to the problems they define. For example, consider some formula of $\mathrm{HGL}(\mathsf{P}, \mathsf{N})$ of the form $\varphi \vee \psi$. For some modal frame $\mathcal{G}$, it is the case that $\mathcal{G} \models \varphi \vee \psi$ if, and only if, for all valuation functions $\mu : \mathsf{P} \cup \mathsf{N} \to \wp(V)$ and for all points u of $\mathcal{G}$, $(\mathcal{G}, \mu, u) \models \varphi \vee \psi$. However, it happens that $\mathcal{G} \models \varphi$ or $\mathcal{G} \models \psi$ if, and only if, for all valuation functions $\mu : \mathsf{P} \cup \mathsf{N} \to \wp(V)$ and for all points u of $\mathcal{G}$, $(\mathcal{G}, \mu, u) \models \varphi$ or for all valuation functions $\mu : \mathsf{P} \cup \mathsf{N} \to \wp(V)$ and for all points u of $\mathcal{G}$, $(\mathcal{G}, \mu, u) \models \psi$. Thus, it might be that $\mathcal{G} \models \varphi \vee \psi$ but that it is not necessarily the case that $\mathcal{G} \models \varphi$ or $\mathcal{G} \models \psi$. This is because there is a universal quantifications of the valuation functions and the points of $\mathcal{G}$ hidden in the expression $\mathcal{G} \models \varphi \vee \psi$ and in general $\forall x(\varphi \vee \psi) \not\equiv \forall x\varphi \vee \forall x\psi$.

Note that it might be argued that HGL is not very "well behaved" as a logic; for instance, it is not closed under negation. However, there are many logics prevalent in descriptive complexity that are not closed under negation, existential second-order logic being perhaps the most prominent example (see Definition 2.1.22).

Remark 2.3.17. The notation used here differs from that used in [BS09]. Modal frames are usually of the form $\mathcal{G} = \langle V, E \rangle$ (to reflect the fact that HGL deals with graphs, or more precisely digraphs), whereas $\mathcal{F} = \langle W, R \rangle$ was the norm in [BS09] (and most basic modal logic texts, e.g. [MV07]); sets of propositional symbols are usually of the form P, whereas Φ was the norm in [BS09] (Greek letters are reserved here for formulae, valuation

functions and paths in digraphs); sets of nominals are usually of the form N, whereas Ψ was the norm in [BS09]; and valuations are usually of the form μ, whereas $\mathbf{V}$ was the norm in [BS09].

2.3.2.4 Metrics and Sub-classes of Formulae

Whilst it is interesting to see what problems a logic can characterise, it is also interesting to question whether there are strict classes of expressiveness within a logic. For this to be investigated metrics derived from formulae are used to build classes. Hybrid Graph Logic already has two built in metrics: the number of proposition symbols and the number of nominal symbols; it is only the number of such symbols that matters, since there is a straight forward isomorphism between two HGL formulae with different sets of symbols of the same size.

Up until now, the logic $\mathrm{HGL}(\mathsf{P}, \mathsf{N})$ has consisted of all formulae with set of proposition symbols P and set of nominal symbols N, such that the logics $\mathrm{HGL}(\mathsf{P}, \mathsf{N})$ and $\mathrm{HGL}(\mathsf{P}', \mathsf{N}')$ are distinct even when $|\mathsf{P}| = |\mathsf{P}'|$ and $|\mathsf{N}| = |\mathsf{N}'|$ are true. Following from the discussion in the previous paragraph, the following class of formulae is defined:

Definition 2.3.18. The class of formulae $\mathrm{HGL}(c, d)$, where $c \geq 0$ and $d \geq 0$, consists of all formulae from $\mathrm{HGL}(\mathsf{P}, \mathsf{N})$, where $c = |\mathsf{P}|$ and $d = |\mathsf{N}|$. Observe that it is the case that both $\mathrm{HGL}(c, d) \subseteq \mathrm{HGL}(c + 1, d)$ and $\mathrm{HGL}(c, d) \subseteq \mathrm{HGL}(c, d + 1)$ hold.

Another measure that can be applied to formulae in Hybrid Graph Logic is of their *quantifier rank* (commonly called *modal depth* in other literature). This is effectively the modal logic version of Definition 2.1.14. Using this measure along with the number of proposition symbols and nominals, a rich taxonomy of Hybrid Graph Logic formulae can be developed.

Definition 2.3.19. Every atomic formula φ has quantifier-rank 0 and it is written that $\mathrm{qr}(\varphi) = 0$ for such formulae.

- If φ is of the form $\neg\psi$ then φ has quantifier-rank defined as $\mathrm{qr}(\varphi) = \mathrm{qr}(\psi)$.
- If φ is of the form $\psi \Rightarrow \psi'$, $\psi \wedge \psi'$ or $\psi \vee \psi'$ then φ has quantifier-rank defined as $\mathrm{qr}(\varphi) = \max\{\mathrm{qr}(\psi), \mathrm{qr}(\psi')\}$.
- If φ is of the form $\Diamond\psi$, $\Diamond^+\psi$, $\Box\psi$, $\Box^+\psi$ or $@_n\psi$ then φ has quantifier-rank defined as $\mathrm{qr}(\psi) + 1$.

Definition 2.3.20. Let c, d and r be non-negative integers, let P be a set of propositional symbols and let N be a set of nominals. The logic $\mathrm{HGL}_r(\mathsf{P}, \mathsf{N})$ consists of those formulae of $\mathrm{HGL}(\mathsf{P}, \mathsf{N})$ of quantifier-rank at most r. The logic $\mathrm{HGL}_r(c, d)$ is the fragment of Hybrid Graph Logic consisting of all those formulae of quantifier-rank at most r in which there are at most c proposition symbols and d nominals.

The logic $\mathrm{HGL}_r(c, d)$ is equated with the class of problems definable by the formulae of that logic; consequently it is written, for example, $\mathrm{HGL}_r(c, d) \subseteq \mathrm{HGL}_{r'}(c', d')$ to denote that any problem definable by a formula of the logic $\mathrm{HGL}_r(c, d)$ can also be defined by a formula of the logic $\mathrm{HGL}_{r'}(c', d')$, and it is written $\mathrm{HGL}_r(c, d) \subset \mathrm{HGL}_{r'}(c', d')$ to denote that $\mathrm{HGL}_r(c, d) \subseteq \mathrm{HGL}_{r'}(c', d')$ but there are problems definable in $\mathrm{HGL}_{r'}(c', d')$ that are not definable in $\mathrm{HGL}_r(c, d)$.

2.3.2.5 Logical Equivalence

Related to the logic $\mathrm{HGL}_r(c,d)$ are various notions of equivalence (see Section 2.1.1.2 for more details).

Definition 2.3.21. Let P be a set of proposition symbols and let N be a set of nominals.

- Let $(\mathcal{G},\mu,u)$ and $(\mathcal{H},\lambda,v)$ be pointed $\mathsf{P}\cup\mathsf{N}$-structures. $(\mathcal{G},\mu,u)$ and $(\mathcal{H},\lambda,v)$ are $(\mathsf{P},\mathsf{N},r)$-*equivalent*, written $(\mathcal{G},\mu,u) \equiv_{\mathrm{HGL}_r(\mathsf{P},\mathsf{N})} (\mathcal{H},\lambda,v)$, if for all formulae φ of $\mathrm{HGL}_r(\mathsf{P},\mathsf{N})$, $(\mathcal{G},\mu,u) \models \varphi$ if, and only if, $(\mathcal{H},\lambda,v) \models \varphi$.
- The $\mathsf{P}\cup\mathsf{N}$-structures $\langle\mathcal{G},\mu\rangle$ and $\langle\mathcal{H},\lambda\rangle$ are $(\mathsf{P},\mathsf{N},r)$-*equivalent*, written $\langle\mathcal{G},\mu\rangle \equiv_{\mathrm{HGL}_r(\mathsf{P},\mathsf{N})} \langle\mathcal{H},\lambda\rangle$, if $(\langle\mathcal{G},\mu\rangle,u) \equiv_{\mathrm{HGL}_r(\mathsf{P},\mathsf{N})} (\langle\mathcal{H},\lambda\rangle,v)$, for all points u and v of $\mathcal{G}$ and $\mathcal{H}$, respectively.
- The modal frames $\mathcal{G}$ and $\mathcal{H}$ are (c,d,r)-*equivalent*, written $\mathcal{G} \equiv_{\mathrm{HGL}_r(c,d)} \mathcal{H}$, if for all formulae φ of $\mathrm{HGL}_r(c,d)$, $\mathcal{G} \models \varphi$ if, and only if, $\mathcal{H} \models \varphi$.

2.3.3 Bisimulation

Basic modal logic is limited in what properties it can express (on finite structures) due to the fact that it is bisimulation-invariant, that is no formula in basic modal logic can differentiate between two given points in two different models for which a valid bisimulation has been given. A bisimulation is defined as follows:

Definition 2.3.22. Given two P-structures $\langle\mathcal{G},\mu\rangle$ and $\langle\mathcal{H},\lambda\rangle$, a non-empty relation $Z \subseteq V^{\mathcal{G}} \times V^{\mathcal{H}}$ is a bisimulation between $\langle\mathcal{G},\mu\rangle$ and $\langle\mathcal{H},\lambda\rangle$ if the following conditions hold for any pair of states $(s,t) \in Z$:

1. For all $p_i \in \mathsf{P}$, $s \in \mu(p_i)$ iff $t \in \lambda(p_i)$ *(base).*
2. For all $u \in V^{\mathcal{G}}$ such that $E^{\mathcal{G}}(s,u)$ holds, there exists some $v \in V^{\mathcal{H}}$ such that $E^{\mathcal{H}}(t,v)$ holds and $(u,v) \in Z$ *(forth).*
3. For all $v \in V^{\mathcal{H}}$ such that $E^{\mathcal{H}}(t,v)$ holds, there exists some $u \in V^{\mathcal{G}}$ such that $E^{\mathcal{G}}(s,u)$ holds and $(u,v) \in Z$ *(back).*

The points $u \in V^{\mathcal{G}}$ and $v \in V^{\mathcal{H}}$ are bisimilar if such a bisimulation Z exists and $(u,v) \in Z$.

A bisimulation is a kind of structural logical equivalence (see Section 2.3.2.5) in the sense that two pointed models $(\mathcal{G},\mu,s)$ and $(\mathcal{H},\lambda,t)$ are bisimilar if, and only if, they agree on the same basic modal logic formulae.

Theorem 2.3.23. *For two pointed* P*-structures* $(\mathcal{G},\mu,s)$ *and* $(\mathcal{H},\lambda,t)$*, the following are equivalent:*

1. *Points* s *and* t *are bisimilar in the models* $\langle\mathcal{G},\mu\rangle$ *and* $\langle\mathcal{H},\lambda\rangle$*, respectively.*
2. $(\mathcal{G},\mu,s) \equiv_{\mathrm{ML}} (\mathcal{H},\lambda,t)$*.*
3. *For any formula* $\varphi \in \mathrm{ML}(\mathsf{P})$ *it holds that* $(\mathcal{G},\mu,s) \models \varphi$ *iff* $(\mathcal{H},\lambda,t) \models \varphi$*.*

Bisimulation is what is known as an algebraic characterisation of the limits of expressiveness of a logic, similar to the algebraic characterisation of first-order logic by Fraïssé [Fra54]. When dealing with finite models it is often easier to think of this characterisation as a game, as is done for first-order and second-order logic in Section 2.1.5.

Definition 2.3.24. The bisimulation game $G_r^b((\mathcal{G}, \mu, s), (\mathcal{H}, \lambda, t))$ is played by two players called Spoiler and Duplicator on the P-structures $\mathcal{A} = \langle \mathcal{G}, \mu \rangle$ and $\mathcal{B} = \langle \mathcal{H}, \lambda \rangle$, and it lasts for r rounds of play. Initially it starts with the pebbles a and b placed on the points s in $\langle \mathcal{G}, \mu \rangle$ and t in $\langle \mathcal{H}, \lambda \rangle$; these are the positions a_0 and b_0. In each round of play, the Spoiler chooses a structure, either $\langle \mathcal{G}, \mu \rangle$ or $\langle \mathcal{H}, \lambda \rangle$ and moves the corresponding pebble, either a or b, onto an adjacent point in the structure. The Duplicator then responds by moving the pebble in the other structure onto an adjacent point. Let the current positions of the pebbles be a_i and b_i, and the new positions a_{i+1} and b_{i+1}. It must be the case that both $E^{\mathcal{G}}(a_i, a_{i+1})$ and $E^{\mathcal{H}}(b_i, b_{i+1})$ hold.

During a play in order for the Duplicator to win it must be the case that at each round for all $p_i \in \mathsf{P}$, $a_i \in \mu(p_i)$ iff $b_i \in \lambda(p_i)$, otherwise the Spoiler wins.

This game and hence bisimulation are only the same as structural equivalence in basic modal logic; in Hybrid Graph Logic Theorem 2.3.23 does not hold. An Ehrenfeucht-Fraïssé style game for bisimulation can be found in most modal logic literature, e.g. [MV07].

Chapter 3

Descriptive Complexity of Optimisation Problems

When describing computational problems it is common to do so using natural language, whereas when describing an implementation of an algorithm on a particular machine, it is common to use a formal notation that explicitly defines the functionality of the machine. Often the language describing the implementation is very technical and it is hard to trace it back up to the natural language of the original problem.

Looking for ways of describing the implementations of algorithms from the field of Finite Model Theory gives the following advantages:

- Ability to reason about computational problems and their implementations using the rich toolset that mathematical model theory and logic provide.
- The natural complexity classes are cleanly captured.
- A different perspective on what makes a problem computationally intractable.

This application of Finite Model Theory to describing algorithms for solving computational problems comes under the broad term of Descriptive Complexity, which is discussed in Section 2.1.4, along with key results from the field in Section 2.1.4.3. The vast majority of research carried out to date in Descriptive Complexity has been concerned with the description of algorithms for solving decision problems and therefore the characterisation of classes of decision problems. It would appear that this has mainly happened because the formal languages used to describe the problems are derived from Boolean logic, which lends itself well to decision problems as a result of the mapping from *true* and *false* to *yes* and *no*.

As the field of Descriptive Complexity is now very well defined and includes a healthy set of results for decision problems, it follows to pose the question: what about optimisation problems? It is not entirely obvious how it could be possible to use a formal language based on Boolean logic to come to an answer that is a number, as opposed to just a Boolean value. The techniques needed are clearly going to involve counting of some sort, but how exactly can a logic "count"? There is also the question of how exactly the numbers are going to be encoded using a finite set of symbols and what the limitations of using this finite set will impose on the types of problems that can be described.

All these questions and more shall be dealt with in this chapter. Whilst they are relatively open ended in the sense that there are essentially an unlimited number of ways of describing optimisation problems using formal languages (in the same way that there are essentially an unlimited number of different computer programming languages) by sticking to the following core theme, it is argued that the research has produced useful results. The core theme of this chapter's research is summarised in the following two questions:

> *What additional linguistic features does a logic that already characterises a class of decision problems need in order to characterise a similar class of optimisation problems?*

and

> *What is the computational complexity of these additional linguistic features in isolation?*

The first question highlights the theme's desire to build upon work that has gone before and the second its desire to analyse the impact of the new results obtained.

3.1 Chapter Outline

The general definitions and notations that this chapter uses and refers to can be found in Section 2.1; in particular the results from Descriptive Complexity in Section 2.1.4 shall be drawn on. The definitions and notations specific to optimisation problems can be found in Section 2.2, the whole of which is applicable to this chapter.

The first section (3.2) looks at past work on characterising optimisation problems; starting with the works covering NP-optimisation problems (3.2.1) before moving onto P-optimisation problems (3.2.2). The next section (3.3) examines some of the short comings with the proposed frameworks for characterising P-optimisation problems before proving that certain approaches cannot ever work unless $\mathsf{P} = \mathsf{NP}$. Following on from this a section (3.4) proposes and proves a framework for characterising P-optimisation problems. Next some examples of using this framework to describe several optimisation problems are presented (3.5) before introducing another framework (3.6), proving that it characterises the class of P-optimisation problems. Finally, the extensions of frameworks to unbounded optimization problems is examined and discussed (3.7).

3.2 Past Research

In this section, the past research into characterising optimisation problems is presented. First in Section 3.2.1 the logical frameworks for characterising classes of non-deterministic polynomial-time optimisation problems are given from [PY91, PR93, KT94, KT95]. Furthermore, some classification results are given from [KT94, KT95]. Secondly in Section 3.2.2 the logical frameworks for characterising classes of deterministic polynomial-time optimisation problems are given from [BM08, Man08].

3.2.1 Characterizing NP_{opt}^{PB}

The first paper that tried to characterise NP-optimisation problems in terms of logic was written by Papadimitriou and Yannakakis [PY91]. They approached the problem by extending Fagin's framework for NP decision problems [Fag74] to the domain of optimisation problems.

They observed that due to Fagin any NP decision problem can be written in the general form given by the formula in Section 2.2.1 and then modified this to create a framework within which optimisation variants of NP decision problems could be described:

$$\mathrm{opt}_{\mathcal{Q}}(\mathcal{A}) = \max_{\mathcal{B}}\{|\{\bar{x} \in A^k \mid (\mathcal{A},\mathcal{B}) \models \exists \bar{y}\, \psi(\bar{x},\bar{y},\bar{S})\}|\} \tag{3.1}$$

where k is the arity of the tuple $\bar{x}$ and the structure $\mathcal{B}$ is a realisation of the relation symbols in the m-tuple $\bar{S}$ over the universe A; the structure $\mathcal{B}$ has vocabulary $\sigma_S = \langle S_1, ..., S_m \rangle$ and is loosely referred to as the *feasible solution* (see Definition 2.2.2 for the exact meaning of a feasible solution).

So instead of finding some solution $\mathcal{B}$ that satisfies $\exists \bar{y}\, \psi$ for all $\bar{x}$, the framework finds the maximum number of different $\bar{x}$ tuples that satisfy $\exists \bar{y}\, \psi$ for each possible solution $\mathcal{B}$. It can be seen that this is the case since the members of the inner set are the $\bar{x}$ tuples that satisfy the formula $\exists \bar{y}\, \psi$ for a particular assignment to the relations in $\bar{S}$ and the outer set's members are the cardinalities of the inner set for each possible assignment to $\bar{S}$. From this set of cardinalities, the largest is the optimal value.

Papadimitriou and Yannakakis named the class of problems that can be described using this framework as MAX NP. They also defined the class MAX SNP as the problems that can be defined in the framework but without the first-order existential component i.e., of the form $\psi(\bar{x},\bar{S})$ rather than $\exists \bar{y}\, \psi(\bar{x},\bar{y},\bar{S})$. The paper [PY91] then shows that some natural problems in NP_{opt}, such as Max-Sat (see Definition 2.1.70), can be defined using the frameworks representing the classes MAX NP and MAX SNP, and that using an approximation preserving reduction, the class MAX SNP has complete problems. They argued that while MAX NP might not capture the class NP_{opt}, since the framework was derived from Fagin's Theorem (see 2.1.75) and captures several natural problems, it is a sound way of describing optimisation problems.

The next key result for the characterisation of optimisation problems comes from a paper by Panconesi and Ranjan in which they show that the class MAX NP is not expressive enough to contain the optimisation variant of the NP problem Max-Clique [PR93]; they then go on to propose a more expressive class of optimisation problems which they call MAX Π_1. The paper is also the first to mention the formal definition of an optimisation problem; it is practically identical to that given in Definition 2.2.2 but without the relationship between NP-optimisation and decision problems.

The proof that the class MAX NP is not large enough to capture all NP-optimisation problems, as given in [PR93], is motivated by the fact that:

$$[\mathcal{A} \models \exists \bar{x} \psi\,(\bar{x}) \wedge \mathcal{A} \subseteq \mathcal{B}] \Rightarrow [\mathcal{B} \models \exists \bar{x}\, \psi(\bar{x})] \tag{3.2}$$

which says that given some formula φ in Σ_1 form that is satisfied by some structure $\mathcal{A}$,

the same formula will hold true on any structure $\mathcal{B}$ that is an extension of $\mathcal{A}$.

Theorem 3.2.1. *The optimisation problem* Max-Clique *cannot be described by the framework in Equation 3.1 and so is not in the class* MAX NP.

Proof. Assume that there exists a formula that characterises the problem Max-Clique in the framework described in Equation 3.1 over graph structures of vocabulary σ_G (see Definition 2.1.5). Call this formula φ and without loss of generality fix the arities of $\bar{x}$, $\bar{y}$ and the solution vocabulary τ. Now, for some graph $\mathcal{G}_1$ assume that the solution structure $\mathcal{A}_1$ is optimal and that:

$$\mathrm{opt}_{MC}(\mathcal{G}_1) = |\{\bar{x} \mid (\mathcal{G}_1, \mathcal{A}_1) \models \exists \bar{y}\, \varphi\}|$$

Take the graph $\mathcal{G}_2$ to be an isomorphic but disjoint copy of $\mathcal{G}_1$ and the solution structure $\mathcal{A}_2$ to be a copy of $\mathcal{A}_1$ under the same isomorphic mapping. Clearly $\mathrm{opt}_{MC}(\mathcal{G}_1) = \mathrm{opt}_{MC}(\mathcal{G}_2)$.

Now create the graph structure $\mathcal{H} = \mathcal{G}_1 \cup \mathcal{G}_2$ and observe that due to the fact that the truth of existential formulae is preserved under extension (Equation 3.2), for some tuple $\bar{a}$, if $(\mathcal{G}_1, \mathcal{A}_1) \models \exists \bar{y}\, \varphi(\bar{a})$ then $(\mathcal{H}, \mathcal{B}) \models \exists \bar{y}\, \varphi(\bar{a})$ and similarly if $(\mathcal{G}_2, \mathcal{A}_2) \models \exists \bar{y}\, \varphi(\bar{a})$ then $(\mathcal{H}, \mathcal{B}) \models \exists \bar{y}\, \varphi(\bar{a})$. Hence the solution structure $\mathcal{B}$ witnesses the following inequality about the optimal value of $\mathcal{H}$:

$$\mathrm{opt}_{MC}(\mathcal{H}) \geq |\{\bar{x} \mid (\mathcal{G}_1, \mathcal{A}_1) \models \exists \bar{y} \varphi\} \cup \{\bar{x} \mid (\mathcal{G}_2, \mathcal{A}_2) \models \exists \bar{y} \varphi\}|$$

And since the pairs of structures $(\mathcal{G}_1, \mathcal{A}_1)$ and $(\mathcal{G}_2, \mathcal{A}_2)$ are disjoint,

$$\mathrm{opt}_{MC}(\mathcal{H}) \geq 2 \cdot \mathrm{opt}_{MC}(\mathcal{G}_1)$$

Which contradicts the fact that the maximum clique in $\mathcal{H}$ is equal to the maximum clique in $\mathcal{G}_1$ (or $\mathcal{G}_2$), since $\mathcal{G}_1$ is not connected to $\mathcal{G}_2$ in $\mathcal{H}$. □

Corollary 3.2.2. MAX NP $\subset$ NP$_{opt}$

Proof. The optimisation problem Max-Clique is an NP-optimisation problem by Definition 2.2.2, since its decision variant is in NP. The proof is direct from this fact and the result in Theorem 3.2.1. □

Panconesi and Ranjan showed that Max-Clique can be written using the following framework:

$$\mathrm{opt}_{\mathcal{Q}}(\mathcal{A}) = \max_{\mathcal{B}} \{|\{\bar{x} \in A^k \mid (\mathcal{A}, \mathcal{B}) \models \forall \bar{y}\, \psi(\bar{x}, \bar{y}, \bar{S})\}|\} \tag{3.3}$$

By giving the formula:

$$\mathrm{opt}_{MC}(\mathcal{G}) = \max_{\mathcal{C}} \{|\{v \mid (\mathcal{G}, \mathcal{C}) \models C(v) \wedge \forall x \forall y\, (C(x) \wedge C(y) \Rightarrow E(x,y) \vee x = y)\}|\|\} \tag{3.4}$$

where the input structure $\mathcal{G}$ is of the vocabulary of graphs (see Definition 2.1.5); structure $\mathcal{C}$ has the vocabulary $\langle C^1 \rangle$, (that is, it contains one unary relation C) and is over the same universe as $\mathcal{G}$.

They called the class of problems that can be described using the framework in Equation 3.3 as MAX Π_1 and exhibit some more example problems within it, but they do not show that any of the problems Papadimitriou and Yannakakis placed in MAX NP are also in MAX Π_1.

Remark 3.2.3. There are some differences between the notation used for the frameworks in the past research [PY91, PR93, KT94, KT95, BM08, Man08] and that used in this chapter (as well as differences between each and every paper). Usually there is no obvious difference between the structure being maximised over and the relation symbols contained within it. By making the difference explicit, using say $\mathcal{B}$ to denote the structure and $S_0, ..., S_m$ to denote the symbols it is clear whether an actual instance or just the abstract symbols are being referred to. In a framework, generally, the symbols are used in the formula and the structure in the mechanics for working out the optimum value.

The other difference is that the structure on which a formula is being evaluated is explicitly stated, as in $(\mathcal{A}, \mathcal{B}) \models \varphi$. Without this notation is has to be implicitly derived as to which symbols in φ are free and which are bound.

It is argued that this more explicit style of notation is necessary since this chapter compares and contrasts many different frameworks, each originally presented in its own style and with its own notation.

The only difference between the frameworks defined by Equations 3.1 & 3.3 is that their first-order components are respectively existential and universal. In a journal paper [KT94], Kolaitis and Thakur allowed the first-order component to be any formula (in prenex normal form) that has free variables from $\bar{S}$ and $\bar{x}$. With this generalisation of the framework they showed that it is equal to the class of *polynomially-bounded* NP-optimisation problems (see Definition 2.2.8) and that there is a proper hierarchy between restrictions of the framework [KT94]. In what follows, both these key results are formally stated.

When restricting the first-order component of the framework, the naming convention follows the quantifier order of this part of the formula. For example, it is a first-order existential formula in the framework in Equation 3.1, giving rise to the name MAX Σ_1 (since it is a framework for maximisation problems). With this new naming scheme it can be seen that MAX Σ_1 = MAX NP and MAX Σ_0 = MAX SNP for the syntactic classes in [PY91], and that the class MAX Π_1 in [PR93] remains unchanged.

Theorem 3.2.4 (Kolaitis and Thakur [KT94])**.** *Let $\mathcal{Q} = (I_\mathcal{Q}, F_\mathcal{Q}, \text{cost}_\mathcal{Q}, \mu)$ be an optimisation problem over vocabulary σ. The following are equivalent:*

1. *$\mathcal{Q}$ is a polynomially-bounded* NP*-optimisation problem, i.e., $\mathcal{Q} \in \text{NP}_{opt}^{PB}$.*
2. *There exists a signature $\tau = \langle \bar{S} \rangle$, consisting solely of relation symbols and disjoint from σ, and a first-order formula $\varphi(\bar{x})$ over $\sigma \cup \tau$, where $\bar{x}$ is a k-tuple of variables, for some k, such that for every instance $\mathcal{A} \in I_\mathcal{Q}$:*

$$\text{opt}_\mathcal{Q}(\mathcal{A}) = \mu_{\mathcal{B}} \{ | \{ \bar{x} \in A^k \mid (\mathcal{A}, \mathcal{B}) \models \varphi(\bar{x}) \} | \} \tag{3.5}$$

   where $\mathcal{B}$ ranges over all τ-structures of universe A.

Proof. If an optimisation problem $\mathcal{Q}$ is definable in the above framework, then the inner set contains the number of $\bar{x}$ tuples that satisfy the formula φ for some particular structure $\mathcal{B}$. Since all $\bar{x}$ are from A^k and $|A^k| = |A|^k$, then $\text{opt}_{\mathcal{Q}}(\mathcal{A}) \leq |A|^k$ and is by Definition 2.2.8 polynomially-bounded.

Now assume that $\mathcal{Q}$ is a polynomially bounded optimisation problem with input structure $\mathcal{A}$ over vocabulary τ. Let k be a positive integer such that for any $\mathcal{A}$ it holds that $\text{opt}_{\mathcal{Q}}(\mathcal{A}) \leq |A|^k$. A new relation R of arity k is now introduced and the following class of problems is defined when $\mu = \max$:

$$K_1 := \{(\mathcal{A}, R) \mid \mathcal{A} \in I_{\mathcal{Q}}, R \subseteq A^k, \text{there is an } \mathcal{B} \in F_{\mathcal{Q}}(\mathcal{A}) \text{ s.t. } \text{cost}(\mathcal{A}, \mathcal{B}) \geq |R|\}$$

and when $\mu = \min$:

$$K_1 := \{(\mathcal{A}, R) \mid \mathcal{A} \in I_{\mathcal{Q}}, R \subseteq A^k, \text{there is an } \mathcal{B} \in F_{\mathcal{Q}}(\mathcal{A}) \text{ s.t. } \text{cost}(\mathcal{A}, \mathcal{B}) \leq |R|\}$$

It can be seen that the class K_1 corresponds to the decision variant of the optimisation problem $\mathcal{Q}$, as defined in 2.2.2 part (vi), and thus is acceptable in NPTIME. By Fagin's seminal characterisation of NP (see Theorem 2.1.75), there exists a second-order existential formula $\exists \bar{S}\, \psi$ of vocabulary $\tau \cup \langle R^k \rangle$ and with ψ being first-order, s.t.:

$$(\mathcal{A}, R) \in K_1 \Leftrightarrow (\mathcal{A}, R) \models \exists \bar{S} \psi$$

With this, it can be seen that finding the optimal solution to the problem $\mathcal{Q}$ is equivalent to finding some extended structure $(\mathcal{A}, R)$ s.t. $\text{opt}_{\mathcal{Q}}(\mathcal{A}) = |R|$ and thus:

$$\text{opt}_{\mathcal{Q}}(\mathcal{A}) = \underset{(R, \mathcal{B}^*)}{\mu} \{|R| \mid (\mathcal{A}, R, \mathcal{B}^*) \models \psi\}$$

Where $\mathcal{B}^*$ ranges over all structures with vocabulary $\sigma = \langle \bar{S} \rangle$ and so ψ is appropriate to the vocabulary $\tau \cup \langle R^k \rangle \cup \sigma$.

In the case that $\mu = \max$ this leads to:

$$\text{opt}_{\mathcal{Q}}(\mathcal{A}) = \max_{(R, \mathcal{B}^*)} \{|\{\bar{x} \in A^k \mid (\mathcal{A}, R, \mathcal{B}^*) \models R(\bar{x}) \wedge \psi\}|\} \tag{3.6}$$

and in the case $\mu = \min$ to:

$$\text{opt}_{\mathcal{Q}}(\mathcal{A}) = \min_{(R, \mathcal{B}^*)} \{|\{\bar{x} \in A^k \mid (\mathcal{A}, R, \mathcal{B}^*) \models \psi \Rightarrow R(\bar{x})\}|\} \tag{3.7}$$

Which are both in the form required by Equation 3.5, with the solution vocabulary being $\langle R^k, \bar{S} \rangle$.

(The proof is based on the one presented in Chapter 10 of [EF99]). □

As was observed in Equation 2.2, the first-order component of a sentence describing an NP decision problem need not be more complex than Π_2; this is also true for the first-order component of Equation 3.6.

Corollary 3.2.5. $\text{NP}^{PB}_{max} = \text{MAX } \Pi_2$.

The differences between the maximisation and minimisation results in the proof of Theorem 3.2.4 is due to the fact that $R(\bar{x}) \wedge \psi(R, \bar{S}^*)$ is inappropriate for minimisation as the value zero (when R is empty) would always be the minimum. Since $\psi(R, \bar{S}^*) \Rightarrow R(\bar{x})$ can be rewritten as $\neg\psi(R, \bar{S}^*) \vee R(\bar{x})$ and $\neg\forall\bar{w}\,\varphi \equiv \exists\bar{w}\,\neg\varphi$, the first-order component in Equation 3.7 is equivalent to a Σ_2-formula. This gives us:

Corollary 3.2.6. $\mathsf{NP}^{PB}_{min} = \mathsf{MIN}\ \Sigma_2$.

Kolaitis and Thakur then show that the quantifier complexity of the first-order formulae in the framework form a proper hierarchy; for maximisation this is:

$$\mathsf{MAX}\ \Sigma_0 \subset \mathsf{MAX}\ \Sigma_1 \subset \mathsf{MAX}\ \Pi_1 \subset \mathsf{MAX}\ \Pi_2$$

As shown in Theorem 3.2.4 and given in Corollary 3.2.5, the class MAX Π_2 captures all *polynomially-bounded* NP-optimisation problems. The problem Max-Connected-Component separates the classes MAX Π_2 and MAX Π_1; Max-Clique separates MAX Π_1 and MAX Σ_1 (see 3.2.1); and Max-SAT separates MAX Σ_1 and MAX Σ_0.

The hierarchy for minimization problems is also proper, but only has two levels as opposed to maximisation's four:

$$\mathsf{MIN}\ \Sigma_0 = \mathsf{MIN}\ \Sigma_1 \subset \mathsf{MIN}\ \Pi_1 = \mathsf{MIN}\ \Sigma_2$$

The problem Min-Chromatic-Number separates the hierarchy; it is in MIN Π_1 but not MIN Σ_1.

In their second paper [KT95], Kolaitis and Thakur observed that for many natural problems the following two properties hold:

1. A feasible solution is a tuple of relations that satisfy some first-order formula.
2. The objective function value is the cardinality of one of these relations.

Based on this observation they then proposed a new framework for capturing NP-optimisation problems, which for some NP-optimisation problem $\mathcal{Q}$ and input structure $\mathcal{A}$, is given by the following theorem:

Theorem 3.2.7 (Kolaitis and Thakur [KT95])**.** *Let $\mathcal{Q} = (I_\mathcal{Q}, F_\mathcal{Q}, \mathrm{cost}_\mathcal{Q}, \mu)$ be an optimisation problem over vocabulary σ. The following are equivalent:*

(i) $\mathcal{Q}$ is a polynomially-bounded NP*-optimisation problem, i.e., $\mathcal{Q} \in \mathsf{NP}^{PB}_{opt}$.*

(ii) There exists a signature $\tau = \langle\bar{S}\rangle$, consisting solely of relation symbols and disjoint from σ, and a first-order formula $\varphi(\bar{x})$ over $\sigma \cup \tau$, where $\bar{x}$ is a k-tuple of variables, for some k, such that for every instance $\mathcal{A} \in I_\mathcal{Q}$:

$$opt(\mathcal{A}) = \mu_{\mathcal{B}}\{|S_0| \mid (\mathcal{A}, \mathcal{B}) \models \varphi\} \tag{3.8}$$

where $\mathcal{B}$ ranges over all τ-structures of universe A and S_0 is a particular relation symbol from $\bar{S}$.

Given the proof for the old framework (see Theorem 3.2.4), it is easy to see that this new framework captures the class of NP-optimisation problems, since it is used in the penultimate step of the proof. The final step of the proof for the old framework is different for maximisation and minimisation problems, whereas for the new framework it is not. The new framework is identified with an F, e.g. MAX F Π_2 corresponds to the class of formulas in the new framework where the first-order component is in Π_2 form.

Remark 3.2.8. There is one very important difference between the "tuple counting" frameworks in [PY91, PR93, KT94] and Kolaitis and Thakur's new "relation cardinality" framework in [KT95]. When considering minimisation problems (i.e. $\mu = \min$), the tuple counting frameworks suffer from the problem that when $(\mathcal{A}, \mathcal{B}) \not\models \psi$ (i.e. the solution $\mathcal{B}$ is infeasible) the cardinality of the emptyset is added to the outer set and since $|\emptyset| = 0$ the minimum value in the outer set will always be zero. So when designing a formula to describe a minimisation problem the implementer must ensure that $(\mathcal{A}, \mathcal{B}) \models \psi$ is always true, for all possible combinations of input structures and realisations of the tuple of relations (unless of course, the minimum value is in fact zero).

It should be noted that relation cardinality style frameworks do not suffer from this "zero cardinality" problem for minimisation problems. A realisation of S_0 will only be added to the outer set if the realisation $\mathcal{B}$ of the feasible solution $\bar{S}$ (which includes S_0) satisfies the formula φ. If there are no valid feasible solution structures $\mathcal{B}$, to an input instance $\mathcal{A}$, the set will be empty and in accordance with Definition 2.2.5 the optimal value is undefined, signifying the absence of an optimal solution.

The relationships between the relation cardinality and tuple counting frameworks for both maximisation and minimisation problems extends the quantifier complexity hierarchy as given in the below theorem:

Theorem 3.2.9 (Kolaitis and Thakur [KT95])**.**

$$\left.\begin{array}{l}\mathsf{MAX}\ \Sigma_0\\ \mathsf{MAX\ F}\ \Sigma_1\end{array}\right\} \subset \mathsf{MAX}\ \Sigma_1 \subset \mathsf{MAX}\ \Pi_1 = \mathsf{MAX\ F}\ \Pi_1 = \mathsf{MAX}\ \Sigma_2 = \mathsf{MAX\ F}\ \Sigma_2 \subset \mathsf{MAX}\ \Pi_2 = \mathsf{MAX\ F}\ \Pi_2 = \mathsf{NP}^{PB}_{max}.$$

$$\left.\begin{array}{l}\mathsf{MIN}\ \Sigma_0 = \mathsf{MIN}\ \Sigma_1 = \mathsf{MIN\ F}\ \Pi_1\\ \qquad\qquad\ \mathsf{MIN\ F}\ \Sigma_1\end{array}\right\} \subset \mathsf{MIN\ F}\ \Sigma_2 \subset \mathsf{MIN\ F}\ \Pi_2 = \mathsf{MIN}\ \Pi_1 = \mathsf{MIN}\ \Sigma_2 = \mathsf{MIN}\ \Pi_2 = \mathsf{NP}^{PB}_{min}.$$

The bracketing used in the statement of Theorem 3.2.9 is to denote that the classes are incomparable. For example, the classes MAX Σ_0 and MAX F Σ_1 both contain problems that are not in the other class, and so there is no containment between them.

3.2.2 Characterising P^{PB}_{opt}

In section 3.2.1 the evolution of the frameworks for syntactic characterisation of NP-optimisation problems were described. This climaxed with Kolaitis and Thakur's new *relation cardinality* style framework for polynomially-bounded NP-optimisation problems [KT95].

This section defines the restriction of the class NP_{opt} to only optimization problems whose decision variants are in P and examines the search for a framework that syntactically characterises this subclass. The formal definition of a polynomial-time optimisation problem is given in Definition 2.2.11.

It is clear that any problem in the class P_{opt} can be written using the frameworks given in Theorem 3.2.4 Equation 3.5 and Theorem 3.2.7 Equation 3.8, since $\mathsf{P}_{opt}^{PB} \subseteq \mathsf{NP}_{opt}^{PB}$ (see Definition 2.2.11). If the converse were true, that is any problem written in those frameworks was in P_{opt}^{PB}, then $\mathsf{P}_{opt}^{PB} = \mathsf{NP}_{opt}^{PB}$ which implies that $\mathsf{P} = \mathsf{NP}$, since the decision variant of the optimisation problem that solves the framework would be in P and the framework can represent problems with NP-complete decision variants, e.g. MAX-CLIQUE (see Equation 3.4) .

Remarkably little research has been done into logically characterising P_{opt} and what has been done does not really answer any of the questions outlined at the beginning of this chapter.

The research starts with a paper by Manyem in which he proposes a framework for describing polynomial-bounded P-optimisation problems [Man08]. Inspired by the work of Kolaitis and Thakur, Manyem (and subsequently with Bueno in [BM08]) attempted to provide a suitable logical framework to characterize P-maximisation problems and P-minimisation problems. Whereas Kolaitis and Thakur's logical framework had been derived from Fagin's seminal characterisation of NP (see Theorem 2.1.75), Bueno and Manyem looked to take Grädel's characterization of P (see Theorem 2.1.76) as the class of problems definable in a particular fragment of existential second-order logic, as their inspiration.

Below is the framework Bueno and Manyem proposed stated as the theorem that was the ultimate goal of their research:

Theorem 3.2.10. *Let $\mathcal{Q} = (I_{\mathcal{Q}}, F_{\mathcal{Q}}, \text{cost}_{\mathcal{Q}}, \mu)$ is an optimization problem over the vocabulary σ. The following are equivalent:*

(i) $\mathcal{Q}$ is a polynomially-bounded P-optimisation problem, i.e., $\mathcal{Q} \in \mathsf{P}_{opt}^{PB}$.

(ii) There exists a vocabulary $\tau = \langle \bar{S} \rangle$, consisting solely of relation symbols disjoint from σ, and a quantifier-free first-order Horn formula $\eta(\bar{x}, \bar{y})$ over $\sigma \cup \tau$, where $\bar{x}$ is a k-tuple of variables, for some k, such that for every instance $\mathcal{A} \in I_{\mathcal{Q}}$:

$$opt_{\mathcal{Q}}(\mathcal{A}) = \underset{\mathcal{B}}{\mu}\{|\{\bar{x} \in A^k \mid (\mathcal{A}, \mathcal{B}) \models \forall \bar{y}\, \eta\}|\} \tag{3.9}$$

where A is the universe of the structure $\mathcal{A}$ and $\mathcal{B}$ ranges over all τ-structures of universe A. η is a quantifier free first-order formula in conjunctive normal form (CNF) that is comprised of clauses that are Horn with respect to the relation symbols in τ.

The framework in Equation 3.9 is a tuple counting framework of a style similar to the framework presented by Kolaitis and Thakur in Theorem 3.2.4. The key difference is that the first-order formula has been restricted to a universal Horn formula. Therefore, following the current classification scheme for the first-order component, the framework when $\mu = \max$ is called MAX Π_1-Horn and when $\mu = \min$ is called MIN Π_1-Horn. For a

further discussion of Horn clauses see Section 2.1.3.1 and for what it means for a formula to be Horn *with respect to the relation symbols in a vocabulary,* see Definition 2.1.65.

Remark 3.2.11. It should be pointed out that Manyem's definition of a P-optimisation problem in [Man08], and subsequently with Bueno in [BM08], is slightly different from the one given in Definition 2.2.11, for they had an extra condition, namely that a feasible solution witnessing the optimal value for an instance should be computable in time polynomial in the size of the instance. This condition has been dropped from Definition 2.2.11 in order to allow the P-optimisation problems presented in this thesis to be analogous to the NP-optimisation problems of [KT94, PR93, PY91]. Moreover, it is felt that the condition is not intrinsic to the notion of the solution of an optimisation problem; dropping Manyem's additional condition provides for a more appropriate analysis.

So, Remark 3.2.11 points out that Manyem's definition of a P-optimisation problem is at variance with the definition to be expected should one proceed analogously to related research on optimization problems, mentioned above.

Manyem then proceeds to demonstrate that any problem from P_{opt}^{PB} can be written using this framework; this proves the (i) $\Rightarrow$ (ii) direction of Theorem 3.2.10. He treated the results for maximisation and minimisation separately, in Theorems 3 and 10 respectively, mainly so that he could deal with the zero cardinality problem in minimisation problems (see Remark 3.2.8), which he chose to cover by not allowing a minimum solution of zero. This design decision in his framework appears to have been made as it allows a close relationship between the feasible solution from Definition 2.2.11 and the vocabulary τ that the structures $\mathcal{B}$ in Theorem 3.2.10 range over (see Remark 3.2.11 for further discussion on feasible solutions).

No result is shown for the converse direction of the theorem, even when restricted to just maximisation or minimisation problems.

Manyem allowed for the use of a built-in successor relation in the formula η, in Theorem 3.2.10 above, but did not explain how $\eta(\bar{x}, \bar{y})$ might be order-invariant (as per Definition 2.1.28). Manyem attempted to demonstrate the efficacy of his framework by defining within it the problems Max-Unit-Flow and Min-SP (see Examples 2.1.2 and 2.2.1). Unfortunately there are errors in both definitions.

In a subsequent paper that Manyem published in collaboration with Bueno [BM08], it is conjectured that the converse of Theorem 3.2.10, (ii) $\Rightarrow$ (i), is false due to the exponential number of feasible solutions that must be considered for the structure $\mathcal{B}$, but other than stating that they could not design an algorithm to solve the framework, no proof is given.

Below is a reworking of Manyem's proofs from Theorems 3 and 10 in [Man08].

Proposition 3.2.12. *Any optimisation problem in* P_{max}^{PB} *can be defined by a sentence of* MAX Π_2-Horn *and any optimisation problem in* P_{min}^{PB} *can be defined by a sentence of* MIN Π_1-Horn.

Proof. Let $\mathcal{Q} = (I_\mathcal{Q}, F_\mathcal{Q}, \text{cost}, \mu) \in \mathsf{P}_{opt}^{PB}$ have input instances over some vocabulary σ. Let m be such that n^m bounds $opt(\mathcal{A})$, for any instance $\mathcal{A} \in I_\mathcal{Q}$ of size $n = |A|$ (where A is the universe of $\mathcal{A}$). Instances of the decision version of $\mathcal{Q}$, call it $\mathcal{Q}_D$, can be taken to be structures over $\sigma \cup \langle R^m \rangle$, where R is a relation symbol of arity m, by encoding the bound k in an instance of $\mathcal{Q}_D$ as the number of tuples in R. By definition, $\mathcal{Q}_D$ is in P; thus, by

Grädel's theorem (see Theorem 2.1.76), $\mathcal{Q}_D$ can be defined by a sentence of $\exists$SO-Horn of the form:

$$\exists S_1 \exists S_2 \ldots \exists S_t \, \forall y_1 \forall y_2 \ldots \forall y_k \, \varphi,$$

where each S_i is a relation symbol not in the underlying signature σ, each y_j is a (first-order) variable, and φ is a quantifier-free first-order formula over $\sigma \cup \tau$, where $\tau = \langle S_1, S_2, \ldots, S_t \rangle$ and φ is Horn with respect to the relation symbols in τ (see Definition 2.1.65).

Let $\mathcal{A}$ be an instance of $\mathcal{Q}$ from $I_{\mathcal{Q}}$. Suppose that there exists a feasible solution $\mathcal{S} \in F_{\mathcal{Q}}(\mathcal{A})$ such that $\text{cost}(\mathcal{A}, \mathcal{S}) \geq \alpha$ when $\mu = \max$, or $\text{cost}(\mathcal{A}, \mathcal{S}) \leq \alpha$ when $\mu = \min$. Thus, for any relation R over A (the universe of $\mathcal{A}$) consisting of α tuples, $(\mathcal{A}, R) \models \exists S_1 \exists S_2 \ldots \exists S_t \, \forall y_1 \forall y_2 \ldots \forall y_k \, \varphi$. Hence when $\mu = \max$,

$$\max_{(\mathcal{B},R)} \{ |\{ \bar{x} \in A^m \mid (\mathcal{A}, \mathcal{B}, R) \models \forall y_1 \forall y_2 \ldots \forall y_k \, \varphi \wedge R(\bar{x}) \}| \} \geq \alpha$$

and otherwise, when $\mu = \min$,

$$\min_{(\mathcal{B},R)} \{ |\{ \bar{x} \in A^m \mid (\mathcal{A}, \mathcal{B}, R) \models \forall y_1 \forall y_2 \ldots \forall y_k \, \varphi \wedge R(\bar{x}) \}| \} \leq \alpha$$

where $\mathcal{B}$ ranges over all $\langle S_1, S_2, \ldots, S_t \rangle$-structures. Thus the problem $\mathcal{Q}$ has, via its decision variant $\mathcal{Q}_D$, been described using the framework MAX Π_1-Horn or MIN Π_1-Horn, as appropriate. □

Remark 3.2.13. There is a technical problem with the case when $\mu = \min$ in the proof of Proposition 3.2.12. If there are no feasible solutions, i.e $(\mathcal{A}, \mathcal{B}) \not\models \forall y_1 \forall y_2 \ldots \forall y_k \, \varphi$ holds for all structures $\mathcal{B}$, then the minimum value will be zero, which is incorrect. Kolaitis and Thakur deal with this in their framework (see Theorem 3.2.4) by adding tuples to the set when $\psi \Rightarrow R(\bar{x})$, so if for any structure $(\mathcal{A}, \mathcal{B}) \not\models \psi$ then in ranging over all values of R, when $R = A^m$ all the tuples are added ensuring a minimum value of $|A|^m$ when there are no feasible solutions, although this still doesn't quite work as the empty set will get added when $R = \emptyset$. For this reason tuple counting frameworks either have to map the value zero to $\perp$ and ignore it as a valid minimum value, or not allow input instances with no valid feasible solutions.

3.3 Failure of Manyem and Bueno's Proposed Framework

In order for a framework F to characterise a complexity class K it must satisfy two properties:

1. Every problem in K can be written as a formula in the framework F.
2. Every formula expressible using the framework F corresponds to some problem in K.

With respect to Manyem's proposed framework (from Theorem 3.2.10); whereas the first point has been proven to hold (see Proposition 3.2.12), the second point has not. Kolaitis and Thakur's frameworks characterised the complexity class NP_{opt}^{PB} as any formula expressed in them corresponds to a problem in NP_{opt}^{PB}. This property of the frameworks is

potentially easier to show for NP_{opt}^{PB} over P_{opt}^{PB} since every optimisation problem as defined in 2.2.2 is an NP-optimisation problem (see Remark 2.2.3) and so it suffices to show that the framework is an optimisation problem as per Definition 2.2.2 in order for it to have the second property.

3.3.1 Expressing NP-hard Problems In the Maximisation Framework

If a framework designed for expressing optimisation problems from P_{opt}^{PB} can represent an NP-hard problem, or in fact a problem that is complete for NP_{opt}^{PB}, then under the assumption that $\mathsf{P} \neq \mathsf{NP}$ it is the case that $\mathsf{P}_{opt}^{PB} \neq \mathsf{NP}_{opt}^{PB}$, and so any complete problem for NP_{opt}^{PB} is not in P_{opt}^{PB}; any framework that can represent such a problem will never be able to characterise P_{opt}^{PB}, unless of course, it turns out that $\mathsf{P} = \mathsf{NP}$. The following proposition formally explains this concept:

Proposition 3.3.1. *Any framework that can characterize an* NP*-hard problem cannot be a framework that characterizes* P_{opt} *(or any combination of the polynomially-bounded, minimization or maximization sub-classes of* P_{opt}*) under the assumption that* $\mathsf{P} \neq \mathsf{NP}$*.*

Proof. If $\mathsf{P} \neq \mathsf{NP}$ then for every NP-complete problem $\mathcal{Q}$ it is the case that there is no polynomial-time algorithm for solving $\mathcal{Q}$ and so $\mathcal{Q} \notin \mathsf{P}$. Some of these NP-complete problems are decision variants of optimisation problems (e.g. Max-Clique or Min-Vertex-Cover, see Appendices of [GJ79]) and so it follows that $\mathsf{P}_{opt} \neq \mathsf{NP}_{opt}$.

Now, take an NP-hard optimisation problem $\mathcal{H}$. This problem cannot be in P_{opt} since the decision variant of $\mathcal{H}$ is reducible (in polynomial-time) to an NP-complete problem (and hence every NP-complete problem), which by definition of completeness and the initial assumption gives $\mathcal{H} \notin \mathsf{P}_{opt}$. Let $\mathcal{H}$ be a problem the the given framework $\mathcal{F}$ can characterise. It follows that $\mathcal{F}$ cannot characterise P_{opt} since not every problem expressible in $\mathcal{F}$ is in P_{opt}.

This proof can be extended to all the proper sub-classes of optimisation problems (combinations of maximisation or minimisation that are either polynomially-bounded or not) simply by observing that each of these sub-classes of NP_{opt} contains an optimisation problem that has an NP-complete decision variant. □

What follows, is a proof that Manyem's proposed framework (see Theorem 3.2.10) cannot hope to characterise P_{opt}^{PB} under the assumption that $\mathsf{P} \neq \mathsf{NP}$, by demonstrating that the conditions of Proposition 3.3.1 hold for it.

Before stating the main theorem, the optimisation problem Max-Horn-2-SAT, which looks to maximise the number of clauses that a truth assignment could possibly satisfy in a Boolean formula consisting of Horn clauses of at most two literals (see Definition 2.1.70), is introduced and the completeness results that the main theorem uses are given:

Lemma 3.3.2 ([JS87])**.** *The decision problem* k-Max-Horn-2-SAT *is* NP*-complete.*

Proof. By a reduction from the decision problem Min-Vertex-Cover, see [JS87]. □

Corollary 3.3.3. *The optimisation problem* Max-Horn-2-SAT *is in* NP_{opt}^{PB} *and is* NP*-hard.*

Observe that the optimisation problem MAX-HORN-2-SAT is polynomially-bounded as the number of clauses is linear in the size of the universe (in fact the universe contains one symbol for each clause) and hence in NP^{PB}_{opt}. It is also NP-hard, since its decision variant, k-MAX-HORN-2-SAT, is NP-complete (see the definition of NP-hard in Section 2.1.53).

Theorem 3.3.4. *There exists an* NP*-hard maximisation problem* $\mathcal{Q} = (I, F, \text{cost}, \max)$ *such that:* $\mathcal{Q}$ *is over* $\sigma = \langle H, Z\rangle$*, where* H *is a relation symbol of arity* 4 *and* Z *is a constant symbol;* $\tau = \langle P, T\rangle$*, where* P *and* T *are both relation symbols of arity* 1*; and* ψ *is a quantifier free first-order formula over* (σ, τ) *with free variable* y *that is Horn with respect to* τ*, such that for every* $\mathcal{A} \in I$*:*

$$\text{opt}(\mathcal{A}) = \max_{\mathcal{B}}\{|\{w \in A \mid (\mathcal{A}, \mathcal{B}, u) \models \forall x_1 \forall x_2 \forall x_3\, \varphi(w)\}|\}$$

with $\mathcal{B}$ *ranging over all* τ*-structures with domain* A *and* w *detailing a value for the variable* y*.*

Proof. Let I be an instance of MAX-HORN-2-SAT of size n; that is, involving clauses $C_1, C_2, \ldots, C_n$ and Boolean variables $X_1, X_2, \ldots, X_n$. Every clause is of one of the following forms:

(i) $X_i \Rightarrow X_j$

(ii) $X_i \wedge X_j \Rightarrow false$

(iii) $true \Rightarrow X_i$

(iv) $X_i \Rightarrow false$.

Define the input signature $\sigma = \langle H, Z\rangle$, where H is a relation symbol of arity 4 and Z is a constant symbol, and the solution signature $\tau = \langle P, T\rangle$, where P and T are both relation symbols of arity 1. Let $\Phi(\mathcal{I})$ be the σ-structure with domain $\{0, 1, \ldots, n\}$, with the constant $Z = 0$ and with the relation H defined as follows:

(i) if clause C_k of I is of the form $X_i \Rightarrow X_j$ then $(i, 0, j, k) \in H$;

(ii) if clause C_k of I is of the form $X_i \wedge X_j \Rightarrow false$ then $(i, j, 0, k) \in H$;

(iii) if clause C_k of I is of the form $true \Rightarrow X_i$ then $(0, 0, i, k) \in H$;

(iv) if clause C_k of I is of the form $X_i \Rightarrow false$ then $(i, 0, 0, k) \in H$.

This completely defines H.

Define the formula Ψ' over $\sigma \cup \tau$ as

$$\begin{aligned}
&\forall c \forall x \forall y(((c \neq Z \wedge x \neq Z \wedge y \neq Z \wedge H(x, Z, y, c) \wedge P(x) \wedge \neg P(y)) \Rightarrow \neg T(c))\\
&\quad \wedge \quad ((c \neq Z \wedge x \neq Z \wedge y \neq Z \wedge H(x, y, Z, c) \wedge P(x) \wedge P(y)) \Rightarrow \neg T(c))\\
&\quad \wedge \quad ((c \neq Z \wedge x \neq Z \wedge y \neq Z \wedge H(Z, Z, x, c) \wedge \neg P(x)) \Rightarrow \neg T(c))\\
&\quad \wedge \quad ((c \neq Z \wedge x \neq Z \wedge y \neq Z \wedge H(x, Z, Z, c) \wedge P(x)) \Rightarrow \neg T(c))).
\end{aligned}$$

Note that Ψ' is a first-order formula over (σ, τ) and is Horn with respect to τ.

We claim that there exists a truth assignment on $X_1, X_2, \ldots, X_n$ making at least m clauses of I *true* if, and only if, there exist unary relations P and T over $\{0, 1, \ldots, n\}$ such that $(\Phi(I), P, T) \models \Psi'$ and $|T| \geq m$.

Suppose that π is a truth assignment on $X_1, X_2, \ldots, X_n$ making at least m clauses of I *true*. Define

$$P = \{i : 1 \leq i \leq n, \pi(X_i) = true\}$$

and

$$T = \{k : 1 \leq k \leq n, \pi \text{ makes clause } C_k \text{ of } I \; true\}.$$

There are 4 cases to consider.

(i) Suppose that C_k is of the form $X_i \Rightarrow X_j$ and that π makes C_k $false$; so, $\pi(X_i) = true$ and $\pi(X_j) = false$. Hence, $P(i)$ and $\neg P(j)$ hold. Consequently, $H(i, 0, j, k) \wedge P(i) \wedge \neg P(j)$ holds. Furthermore, $\neg T(k)$ holds and so

$$(H(i, 0, j, k) \wedge P(i) \wedge \neg P(j)) \Rightarrow \neg T(k) \text{ holds.}$$

If π makes C_k $true$ then $\pi(X_i) = false$ or $\pi(X_j) = true$, and so at least one of $\neg P(i)$ and $P(j)$ holds with the consequence that

$$(H(i, 0, j, k) \wedge P(i) \wedge \neg P(j)) \Rightarrow \neg T(k) \text{ holds.}$$

(ii) Suppose that C_k is of the form $X_i \wedge X_j \Rightarrow false$ and that π makes C_k $false$; so, $\pi(X_i) = true$ and $\pi(X_j) = true$. Hence, $P(i)$ and $P(j)$ hold. Consequently, $H(i, j, 0, k) \wedge P(i) \wedge P(j)$ holds. Furthermore, $\neg T(k)$ holds and so

$$(H(i, j, 0, k) \wedge P(i) \wedge P(j)) \Rightarrow \neg T(k) \text{ holds.}$$

If π makes C_k $true$ then $\pi(X_i) = false$ or $\pi(X_j) = false$, and so at least one of $\neg P(i)$ and $\neg P(j)$ holds with the consequence that

$$(H(i, j, 0, k) \wedge P(i) \wedge P(j)) \Rightarrow \neg T(k) \text{ holds.}$$

(iii) Suppose that C_k is of the form $true \Rightarrow X_i$ and that π makes C_k $false$; so, $\pi(X_i) = false$. Hence, $\neg P(i)$ holds. Consequently, $H(0, 0, i, k) \wedge \neg P(i)$ holds. Furthermore, $\neg T(k)$ holds and so

$$(H(0, 0, i, k) \wedge \neg P(i)) \Rightarrow \neg T(k) \text{ holds.}$$

If π makes C_k $true$ then $\pi(X_i) = true$, and so $P(i)$ holds with the consequence that

$$(H(0, 0, i, k) \wedge \neg P(i)) \Rightarrow \neg T(k) \text{ holds.}$$

(iv) Suppose that C_k is of the form $X_i \Rightarrow false$ and that π makes C_k $false$; so, $\pi(X_i) = true$. Hence, $P(i)$ holds. Consequently, $H(i, 0, 0, k) \wedge P(i)$ holds. Furthermore,

$\neg T(k)$ holds and so

$$(H(i,0,0,k) \wedge P(i)) \Rightarrow \neg T(k) \text{ holds.}$$

If π makes C_k *true* then $\pi(X_i) = false$, and so $\neg P(i)$ holds with the consequence that

$$(H(i,0,0,k) \wedge P(i)) \Rightarrow \neg T(k) \text{ holds.}$$

Consequently, by definition of H, $(\Phi(I), P, T) \models \Psi'$ and $|T| \geq m$.

Conversely, suppose that there exist unary relations P and T over $\{0, 1, \ldots, n\}$ such that $(\Phi(I), P, T) \models \Psi'$ and $|T| \geq m$. Consider the clause C_k of I. Again, there are 4 cases.

(*i*) Suppose that $k \in T$ and that C_k is of the form $X_i \Rightarrow X_j$; so $H(i,0,j,k)$ holds in $\Phi(I)$. As $(\Phi(I), P, T) \models \Psi'$, we must have that $\neg P(i) \vee P(j)$ holds. Thus, $\pi(X_i) = false$ or $\pi(X_j) = true$ and we have that clause C_k of I is *true* under π.

(*ii*) Suppose that $k \in T$ and that C_k is of the form $X_i \wedge X_j \Rightarrow false$; so $H(i,j,0,k)$ holds in $\Phi(I)$. As $(\Phi(I), P, T) \models \Psi'$, we must have that $\neg P(i) \vee \neg P(j)$ holds. Thus, $\pi(X_i) = false$ or $\pi(X_j) = false$ and we have that clause C_k of I is *true* under π.

(*iii*) Suppose that $k \in T$ and that C_k is of the form $true \Rightarrow X_i$; so $H(0,0,i,k)$ holds in $\Phi(I)$. As $(\Phi(I), P, T) \models \Psi'$, we must have that $P(i)$ holds. Thus, $\pi(X_i) = true$ and we have that clause C_k of I is *true* under π.

(*iv*) Suppose that $k \in T$ and that C_k is of the form $X_i \Rightarrow false$; so $H(i,0,0,k)$ holds in $\Phi(I)$. As $(\Phi(I), P, T) \models \Psi'$, we must have that $\neg P(i)$ holds. Thus, $\pi(X_i) = false$ and we have that clause C_k of I is *true* under π.

Consequently, the truth assignment π makes at least m clauses of I *true*.

By defining Ψ as $\Psi' \wedge T(w)$ and using the result from [JS87] that MAX-HORN-2-SAT is NP-complete, the required result is obtained. □

The above proof has shown that the MAX Π_1-Horn framework can be used to describe an NP-complete problem by demonstrating the existence of an optimisation problem that can solve the decision variant of MAX-HORN-2-SAT. If follows that this optimisation problem must be NP-hard.

3.3.2 Expressing NP-hard Problems in the Minimisation Framework

Now, attention is turned to the minimisation variant of Manyem's framework (MIN Π_1-Horn) and it is considered whether or not an NP-hard problem can be represented in it. As it turns out, the proof of the previous theorem can be adapted to represent the same problem (MAX-HORN-2-SAT) in the MIN Π_1-Horn framework. This is based on the observation that maximising the number of clauses that are true in a Boolean formula is the same as minimising the number of false clauses.

Lemma 3.3.5. *Maximising the number of clauses that are true in a Boolean formula is equivalent to minimising the number of clauses that are false.*

Proof. Note that if a Boolean formula consists of the set of C clauses and under some truth assignment T there are k clauses made true, then there are $|C| - k$ clauses that are made false. Let the maximum number of satisfiable clauses in a formula φ be k and say that the truth assignment T satisfies k clauses in φ. The claim is that the minimum number of unsatisfied clauses is $|C| - k$ and that T witnesses this in φ. Assume that the minimum number of unsatisfied clauses in φ is actually l, where $l < |C| - k$. In this situation all the other clauses are therefore satisfied, so the maximum number of satisfied clauses in φ is $k \geq |C| - l > k$, giving $k > k$ and thus contradicting the claim. □

Theorem 3.3.6. *There exists an* NP*-hard minimisation problem* $\mathcal{Q} = (I, F, \text{cost}, \min)$ *such that:* $\mathcal{Q}$ *is over* $\sigma = \langle H, Z \rangle$*, where* H *is a relation symbol of arity* 4 *and* Z *is a constant symbol;* $\tau = \langle P, T \rangle$*, where* P *and* T *are both relation symbols of arity* 1*; and* ψ *is a quantifier free first-order formula over* (σ, τ) *with free variable* y *that is Horn with respect to* τ*, such that for every* $\mathcal{A} \in I$*:*

$$\text{opt}(\mathcal{A}) = \min_{\mathcal{B}} \{|\{w \in A \mid (\mathcal{A}, \mathcal{B}) \models \forall x_1 \forall x_2 \forall x_3 \varphi(w)\}|\}$$

with $\mathcal{B}$ *ranging over all* τ*-structures with domain* A *and* w *detailing a value for the variable* y*.*

Proof. Let the minimisation problem in question be Min-Horn-Not-2-Sat, which is the problem that given a Boolean formula φ over m variables in conjunctive normal form asks: what is the minimum number of clauses that are not satisfied by any assignment to the m variables in φ? Observe that by Lemma 3.3.5 this problem is equivalent to Max-Horn-2-SAT and thus by Corollary 3.3.3 is also NP-hard.

The proof now proceeds exactly as per that of Theorem 3.3.4 above, except that Ψ is defined as $\Psi' \wedge \neg T(y)$. Note that Ψ still has the Horn property under this modification. Instead of showing that a truth assignment π satisfies at least m clauses if, and only if, $(\Phi(I), P, T) \models \Psi$ and $|T| \geq m$, it is necessary to show that a truth assignment π satisfies *at most* m clauses if, and only if, $(\Phi(I), P, T) \models \Psi$ and $|T| \leq m$; this is because the problem being represented in the framework is a minimization problem rather than a maximisation problem.

The key to showing this relationship is that the cases in the proof of Theorem 3.3.4 cover all possible types of clauses in a Horn-2-SAT formula.

Assume that π satisfies at most m clauses in I and that $|T| > m$. This would mean that there is some clause C_k in T that is *not* satisfied by the truth assignment π. Clause C_k must be of one of the four forms given in the second part of the proof of Theorem 3.3.4 and so it follows that π satisfies C_k, leading to a contradiction.

Assume that $|T| \leq m$ but that π satisfies more than m clauses in I. This means that some clause C_k is satisfied by π but is not in T. Clause C_k must be one of the four forms given in the first part of the proof of Theorem 3.3.4 and so it follows that C_k is in T, leading to a contradiction.

Hence the formula $\Psi' \wedge \neg T(y)$ characterises the minimisation problem, Min-Horn-Not-2-SAT and so the framework MIN Π_1-Horn can represent NP-hard problems. □

3.3.3 Discussion of the results

While the inappropriateness of the framework proposed by Manyem for characterising P^{PB}_{max} was long suspected, and the original motivation of the research was to establish this, the above result for the minimisation variant of the framework was not expected. This is because the algorithm for solving the decision problem Horn-SAT (see Definition 2.1.63) works out the minimum number of literals in a satisfying assignment as a side-effect of applying the algorithm of unit propagation, whereas the problem Min-Horn-SAT is after the minimum number of clauses.

3.4 A Fixed-Point Framework for P^{PB}_{opt}

The rationale in Manyem's work [BM08, Man08], for modifying Kolaitis and Thakur's NP^{PB}_{opt} framework [KT94, KT95], is based on the observation that while NP is captured by sentences in second-order existential logic (Fagin's Theorem [Fag74]), P is captured by restricting the first-order component in second-order sentences to universal Horn formulae [Grä91a]. This led Manyem to simply modify the first-order component of Kolaitis and Thakur's NP^{PB}_{opt} framework to construct a framework for P^{PB}_{opt}.

As was discussed in the previous section, this modification of the framework, while quite understandable, introduces a complexity gap and as such cannot capture both maximisation and minimisation problems. In this section the characterisation of P as the class of problems captured by Immerman's iterative fixed-point logic [Imm82, Var82] is used to construct a framework that captures the complexity class P^{PB}_{opt}. Examples are then given of key problems from this class along with formulae that express them within the framework.

3.4.1 Characterisation of P_{opt}

Manyem's method for capturing polynomially bounded P-optimisation problems [BM08, Man08] is based on a restriction of Kolaitis and Thakur's tuple counting framework that captures NP^{PB}_{max} [KT94]. It is argued that this is an inappropriate approach because:

1. Manyem's framework does not capture P^{PB}_{max} as a solution to the framework is NP-hard (see Theorem 3.3.4).

2. Tuple counting frameworks are not appropriate for minimisation problems, as is argued in [KT95], which led to a more appropriate relation cardinality framework.

3. The distinction between maximisation and minimisation problems in a class is the same as the difference between a class and its complement (as presented in [KT95]). Since $\mathsf{P} = \mathsf{co\text{-}P}$ it appears more appropriate to capture both P^{PB}_{max} and P^{PB}_{min} using exactly the same framework.

To address the above remarks, a framework based on FO(IFP) logic is proposed and examples of problems described using it are given.

Theorem 3.4.1. *A problem $\mathcal{Q}$ is a polynomially bounded P-optimisation problem (i.e. $\mathcal{Q} \in \mathsf{P}^{PB}_{opt}$) iff there exists some formula φ over τ, the vocabulary of $\mathcal{Q}$, such that for any $\mathcal{A} \in I_{\mathcal{Q}}$, where $\mathcal{A}$ is and ordered structure, the optimal value $\mathrm{opt}_{\mathcal{Q}}(\mathcal{A})$ is given by:*

$$\mathrm{opt}_{\mathcal{Q}}(\mathcal{A}) = \mathit{depth}\left([\mathrm{IFP}_{R,\bar{x}}\,\varphi]\,(\bar{t})\right) \tag{3.10}$$

where φ is a formula in the logic FO(IFP) *with free variables consisting only of the k-ary relational symbol $R \notin \tau$ and the k-tuple $\bar{x}$. The k-tuple $\bar{t}$ consists of constant symbols from τ.*

Proof. Suppose that $\mathcal{Q} \in \mathsf{P}^{PB}_{opt}$ and that for some fixed k, $\mathrm{opt}_{\mathcal{Q}}(\mathcal{A}) \leq |A|^k$ for all structures $\mathcal{A} \in I_{\mathcal{Q}}$. Let $W \notin \tau$ be a new k-ary relation symbol and let the binary relation μ be such that if $\mathcal{Q}$ is a maximisation problem then $\mu(x,y) \Leftrightarrow x \geq y$, otherwise if $\mathcal{Q}$ is a minimisation problem then $\mu(x,y) \Leftrightarrow x \leq y$. The class

$$K := \left\{(\mathcal{A}, W) \mid \mathcal{A} \in I_{\mathcal{Q}}, W \subseteq A^k, \text{ there is an } \mathcal{S} \in F_{\mathcal{Q}} \text{ s.t. } \mu\left(\mathrm{cost}(\mathcal{A},\mathcal{S}), |W|\right)\right\}$$

is equivalent to the class of $(\mathcal{A}, c)$ structures that the decision variant of $\mathcal{Q}$ accepts on, with $c = |W|$ and thus the problem of computing whether some $(\mathcal{A}, W)$ structure is in K, is in P. As a consequence of Corollary 2.1.79, there exists some sentence $[\mathrm{IFP}_{S,\bar{x}}\,\psi]\,(\bar{u})$ as per Definition 2.1.39, that characterises K such that

$$(\mathcal{A}, W) \in K \Leftrightarrow (\mathcal{A}, W) \models [\mathrm{IFP}_{S,\bar{y}}\,\psi(W)]\,(\bar{u})$$

where $S \notin \tau$ is an m-ary relation symbol (and hence $\bar{y}$ and $\bar{u}$ are m-tuples) and ψ has free variables $(W, S, \bar{y})$. Note that it is not necessarily the case that $m = k$.

Now, define the following formulae:

$$\begin{aligned} \varphi_{max}(R,\bar{x}) \;=\; & [\mathrm{IFP}_{S,\bar{y}}\,\psi(R)]\,(\bar{u}) \wedge \\ & \left(\bar{x} = \overline{min} \vee \exists \bar{z}\,(R\,(\bar{z}) \wedge \bar{z} \leq \bar{x} \wedge \neg\,\exists \bar{v}\,(\bar{z} < \bar{v} < \bar{x}))\right) \end{aligned} \tag{3.11}$$

$$\begin{aligned} \varphi_{min}(R,\bar{x}) \;=\; & [\mathrm{IFP}_{S,\bar{y}}\,\psi(R)]\,(\bar{u}) \vee \\ & \bar{x} \neq \overline{max} \wedge \left(\bar{x} = \overline{min} \vee \exists \bar{z}\,(R(\bar{z}) \wedge \bar{z} < \bar{x} \wedge \neg\,\exists \bar{v}(\bar{z} < \bar{v} < \bar{x}))\right) \end{aligned} \tag{3.12}$$

If $\mathcal{Q}$ is a maximisation problem then

$$\mathrm{opt}_{\mathcal{Q}}(\mathcal{A}) = \mathit{depth}\left([\mathrm{IFP}_{R,\bar{x}}\,\varphi_{max}]\,(\overline{min})\right)$$

and otherwise (i.e. $\mathcal{Q}$ is a minimisation problem) then

$$\mathrm{opt}_{\mathcal{Q}}(\mathcal{A}) = \mathit{depth}\left([\mathrm{IFP}_{R,\bar{x}}\,\varphi_{min}]\,(\overline{max})\right)$$

both of which are in the appropriate form (see Equation 3.10). All that remains is to prove that the depth of the IFP operators that φ_{max} and φ_{min} give rise to is equal to the optimal value of the problem $\mathcal{Q}$, for which $[\mathrm{IFP}_{S,\bar{y}}\,\psi]\,(\bar{u})$ is an oracle for the decision variant $\mathcal{Q}_D$, over $(\mathcal{A}, W)$ structures. The proof for this proceeds by induction on the sequence of applications of the IFP operator.

First note that the IFP operator defined by the sub-formula

$$\left(\bar{x} = \overline{min} \vee \exists \bar{z} \left(R\left(\bar{z}\right) \wedge \bar{z} < \bar{x} \wedge \neg \exists \bar{v} \left(\bar{z} < \bar{v} < \bar{x}\right)\right)\right)$$

of φ_{max}, constructs a sequence s.t. at each step $|R^i| = i$. This sequence is also constructed in order, starting with the smallest tuple $\overline{min}$ and adding one tuple at each step until $R^{|A|^k} = A^k$, which is a fixed point. For the base case R^1, observe that when $R = \emptyset$ there is no $\bar{z}$ that satisfies the right-hand part of the disjunction, so the only tuple satisfying the formula is $\overline{min}$, giving $R^1 = \{\overline{min}\}$ and hence $|R^1| = 1$. For the inductive step R^{i+1}, assume that R^i contains the first i tuples. The right-hand part of the disjunction is satisfied by any tuple that is the successor of one that is contained in R^i giving

$$R^{i+1} = \{\overline{min}, succ(\overline{min}) + succ(\overline{min+1}), \ldots, succ(\overline{i-1}), succ(\bar{i})\}$$

and hence $|R^{i+1}| = i + 1$, as required. Note that the first tuple in R^{i+1} comes from the left-hand side of the disjunction and that *succ* is used to signify the successor of a tuple (see Defintion 2.1.26).

A similar sub-formula of φ_{min}

$$\bar{x} \neq \overline{max} \wedge \left(\bar{x} = \overline{min} \vee \exists \bar{z} \left(R(\bar{z}) \wedge \bar{z} < \bar{x} \wedge \neg \exists \bar{v} (\bar{z} < \bar{v} < \bar{x})\right)\right)$$

adds to the sub-formula of φ_{max} the condition that a tuple $\bar{x}$ cannot be added if it is the largest tuple, $\overline{max}$. This means that the fixed point is at $R^{|A|^k-1} = A^k \setminus \{\overline{max}\}$. These sub-formulae are the *counting* components of φ_{max} and φ_{min}.

Now consider the case where $\mathcal{Q}$ is a maximisation problem. The optimal value occurs at the largest $|W|$ s.t. $(\mathcal{A}, W) \models [\mathrm{IFP}_{S,\bar{x}}\, \psi]\,(\bar{u})$. The counting component of φ_{max} will keep adding a tuple to R^i as long as $(\mathcal{A}, R^i) \models \left[\mathrm{IFP}_{S,\bar{x}}\, \psi(R^i)\right](\bar{u})$. Say $w = \mathrm{opt}_{\mathcal{Q}}(\mathcal{A})$, then the last point in the sequence to which a tuple is added will be to R^w, giving $R^{w+1} = R^w \cup \{\overline{w+1}\}$. Since $(\mathcal{A}, R^{w+1}) \not\models \left[\mathrm{IFP}_{S,\bar{x}}\, \psi(R^{w+1})\right](\bar{u})$ then R^{w+1} is a fixed-point, giving a depth of w, which is the optimal value.

If there is no solution to $\mathcal{Q}$ for some particular $\mathcal{A} \in I_{\mathcal{Q}}$, i.e. $F(\mathcal{A}) = \emptyset$, the oracle for the decision variant $\mathcal{Q}_D$ will never evaluate to true. This means that $(\mathcal{A}, R^0) \not\models [\mathrm{IFP}_{S,\bar{x}}\, \psi]\,(\bar{u})$ since $R^0 = \emptyset$ and so this will be the fixed-point. Since $\overline{min} \notin \emptyset$ the depth operator will return $\perp$, as required.

For the other case, where $\mathcal{Q}$ is a minimisation problem, the optimal value occurs at the smallest $|W|$ s.t. $(\mathcal{A}, W) \models [\mathrm{IFP}_{S,\bar{x}}\, \psi]\,(\bar{u})$. While the oracle for the decision variant $\mathcal{Q}_D$ evaluates to false, the counting component of φ_{min} will add one tuple (as long as it isn't equal to $\overline{max}$). At the point R^w where the oracle first evaluates to true, the formula φ_{min} is satisfied by any $\bar{x} \in A^k$, including $\overline{max}$ and so giving $R^{w+1} = A^k$. This is a fixed point since it contains all possible tuples. The depth operator returns w, which is as required since $\mathrm{opt}_{\mathcal{Q}}(\mathcal{A}) = |R^w| = w$.

When there is no solution to a minimisation problem, as for a maximisation problem, the oracle for the decision variant will never evaluate to true. This means that only tuples accepted by the counting component of φ_{min} will be added to R, and that the fixed point in the sequence will be $R^{|A|^k-1} = A^k \setminus \{\overline{max}\}$. Since $\overline{max}$ is not an element of this

fixed-point the depth operator will return $\perp$, as required.

Conversely, given a formula in the form of Equation 3.10, the process of computing a solution is in itself an optimisation problem. The decision variant of this problem is one that asks the question: "is the inductive depth of $\varphi(R, \bar{x}) \geq k$?" on $(\mathcal{A}, k)$ structures. This problem is in P, since any machine that constructs R can easily be augmented with a counter that measures the inductive depth of R without adding any degree of additional complexity to the machine. By simply comparing the inductive depth of R with the constant k an answer to the decision problem can be made in polynomial time. Since it is in P, therefore the optimisation problem is in P_{opt}, as required. □

Since FO(IFP) = FO(LFP) (see Theorem 2.1.78) the oracle for the decision variant of an optimisation problem, as used in the proof of Theorem 3.4.1, can also be of the form $[\mathrm{LFP}_{S,\bar{x}}\, \psi]\,(\bar{u})$; using a least fixed-point rather than an inflationary fixed-point operator. The two counting formulae, φ_{max} and φ_{min} (Equations 3.11 and 3.12 respectively) give rise to monotone operators, since the relation R is only referred to positively. Therefore the least fixed-point of these formulae is equal to the inflationary fixed-point and hence throughout the statement and proof of Theorem 3.4.1 IFP can be replaced with LFP, giving rise to the following corollary:

Corollary 3.4.2. *A problem $\mathcal{Q}$ is a polynomially-bounded P-optimisation problem (i.e. $\mathcal{Q} \in \mathsf{P}^{PB}_{opt}$) iff there exists some formula φ over τ, the vocabulary of $\mathcal{Q}$, such that for any $\mathcal{A} \in I_{\mathcal{Q}}$, the optimal value $opt_{\mathcal{Q}}(\mathcal{A})$ is given by:*

$$opt_{\mathcal{Q}}(\mathcal{A}) = depth\left(\left[\mathrm{LFP}_{R,\bar{x}}\, \varphi\right](\bar{t})\right) \tag{3.13}$$

where φ is a formula in the logic FO(LFP) *with free variables consisting only of the k-ary relational symbol $R \notin \tau$ and the k-tuple $\bar{x}$. The k-tuple $\bar{t}$ consists of constants from τ.*

Another observation about the oracle for the decision variant of an optimisation problem, as used in the proof of Theorem 3.4.1, is that it is of the form $[\mathrm{IFP}_{S,\bar{x}}\, \psi]\,(\bar{u})$, where ψ is a formula from the language FO, not FO(IFP). This is due to the absence of a syntactic hierarchy linked to the number of levels of fixed-point operators within FO(IFP/LFP) logics (see Theorem 2.1.36). With this in mind it can immediately be seen that the proof of Theorem 3.4.1 leads to the following corollary:

Corollary 3.4.3. *A formula φ that describes an optimisation problem $\mathcal{Q} \in \mathsf{P}^{PB}_{opt}$ in the framework used in Theorem 3.4.1 need only contain at most one level of fixed-point operators; all sub-formulae of the formula $\varphi \in$* FO(IFP) *that are of the form $[\mathrm{IFP}_{S,\bar{x}}\, \psi]\,(\bar{u})$ have $\psi \in$* FO*.*

As will later be shown in the examples in Section 3.5, some problems (such as Min-SP in Section 3.5.1) require only the outermost fixed-point operator.

3.5 Examples (using the fixed-point framework)

In this section the P^{PB}_{opt} problems introduced in Section 2.2.3 are described using the framework from Theorem 3.4.1.

3.5.1 Shortest Path

The minimisation problem Shortest Path (or Min-SP) is defined in Section 2.2.3.5 and is concerned with finding the shortest path between two vertices in a directed graph. The vocabulary for input instances of this problem is the vocabulary of graphs $\sigma_G = \langle E^2 \rangle$ extended to include two distinct vertices s and t, respectively representing the start and end vertices of the path through the graph. This results in the vocabulary $\sigma_{SP} = \sigma_{Gst} = \sigma_G \cup \langle s, t \rangle = \langle E^2, s, t \rangle$.

Algorithms that compute the length of the shortest path between s and t in polynomial time (e.g. from Chapter 1 of [Pap94]) do so using a breadth-first search technique. Starting from the vertex s they construct all possible *distinct* paths. At each iteration in the algorithm all paths are extended by 1 edge; any branching results in a new path being added and any collisions between paths result in the longest one being deleted. The first path to reach the end vertex t is the shortest path in the graph between s and t. Since at each iteration the algorithm only increases the length of all the paths by 1, the total number of iterations required before t is reached (and the algorithm terminates), is equal to the length of the shortest path between s and t. More formally, at the i^{th} iteration of the algorithm the length of all the current paths is equal to i, giving an equivalence between inductive depth and cost in the problem Shortest Path, which is as required by the framework in Theorem 3.4.1.

As a starting point, examine how the problem Reachability is defined in the logic FO(LFP) (see Chapter 4 of [Imm99]). The problem Reachability asks the question "is there a path between s and t?" Firstly, a first-order formula defining a fixed-point operator that constructs a relation R is given:

$$\psi_{tc}(R, x, y) := \ (x = y) \vee \exists z \left(E\left(z, y\right) \wedge R\left(x, z\right) \right)$$

This formula, ψ_{tc} gives rise to a monotone operator since R appears only positively. The least fixed-point of ψ_{tc} is the same as the transitive closure of the edge relation E and as such, reachability can be defined as:

$$\text{Reachability} \ \equiv \ \mathcal{A} \models \left[\text{LFP}_{R,x,y}\, \psi_{tc}\right](s, t)$$

Where $\mathcal{A}$ is a structure of the vocabulary σ_{SP}. If this operator is visualised constructing the relation R on a graph then it can be seen that after the first iteration R only contains, for all vertices v, the tuples (v, v); that is for all tuples $(u, v) \in R^1$ the distance between u and v is ≤ 0. At the next iteration, for all vertices v and w, if there is an edge between v and w then $(v, w) \in R^2$; hence for all tuples $(u, v) \in R^2$ the distance between u and v is ≤ 1. On subsequent iterations new tuples are added to R if there is an edge (v, w) and $(u, v) \in R$ for some $u \neq w$. If the construction is followed, it can be seen that the presence of the tuple $(u, v) \in R^i$ implies that there is a path from u to v of length $\leq i - 1$, and that the least fixed-point of ψ_{tc} will contain the tuple (s, t) if there is a path from s to t.

The transitive closure of a relation contains all pairs of vertices that are connected. In Min-SP the only vertices of interest are those that are connected to the start and end vertices and more so, the only *paths* of interest are those between s and t. So ψ_{tc} can be restricted to:

$$\psi_1(R,x,y) := \ (x = s \wedge y = s) \vee \exists z \left(E\left(z,y\right) \wedge R\left(x,z\right)\right)$$

Observe that in ψ_1 the tuple (x,y) will only be added to R^{i+1} if $x = s$ or some tuple $(x,z) \in R^i$; this is because the start of all paths is fixed to s, and so ψ_{tc} can be further restricted to only constructing a unary relation:

$$\psi_2(R,x) := \ (x = s) \vee \exists y \left(E(y,x) \wedge R(y)\right)$$

The presence of $x \in R^i$ means that there is a path between s and x of length $\leq i-1$, or more importantly the absence of a vertex ($x \notin R^i$) at the i^{th} iteration means there is no path of length $\leq i-1$ between s and x. Now, if $t \notin R^i$ then there is no path between s and t of length $\leq i-1$ but if $t \in R^{i+1}$ then there is a path between s and t of length $\leq i$, therefore if $t \notin R^i \wedge t \in R^{i+1}$ then the shortest path between s and t is of length i.

In the description of a problem in the framework presented in Theorem 3.4.1, it is required that the optimal value is equal to the inductive depth of the outermost fixed-point operator *minus one.* As shown above, at the point where t is added to R the inductive depth minus one equals the length of the shortest path between s and t, implying that if $t \in R^i$ then $R^i = R^{i+1}$; the iteration where t is added must be the fixed-point. This property can be achieved by adding an extra term to ψ_2, giving:

$$\psi_3(R,x) := \ [x = s \vee \exists y \left(E(y,x) \wedge R(y)\right)] \wedge \neg R(t)$$

Introducing this extra term produces a formula that is not positive in R, and so does not define a monotone operator. This is not a problem though, as the framework in Theorem 3.4.1 uses inflationary fixed-point operators, which do not have this syntactic restriction. All that is required is to consider what happens when s and t are not connected. This is handled by the *depth* operator returning $\perp$ (undefined) when the relation R does not contain some constant; in this case R is required to contain t, giving:

$$\textsc{Min-SP} \ \equiv \ \mathit{depth}\left(\left[\mathrm{IFP}_{R,x}\,\psi_3\right](t)\right)$$

for some structure $\mathcal{A}$ in the vocabulary σ_{SP}.

In order to define Min-SP using least fixed-point rather than inflationary fixed-point operators (and hence describing the problem within the LFP based framework from Corollary 3.4.2), a different method for ensuring that the iteration where t is added is the fixed-point must be used. Rather than stopping any more tuples being added at the point where $t \in R^i$, as formula ψ_3 does with the term $\ldots \wedge \neg R(t)$, the operator must force a fixed-point by adding every element from the universe to R. This is done by adding a term to ψ_2 that accepts any element x if it equals s, is connected to an element already in R *or* if there is some element in R that is connect to t:

$$\psi_4(R,x) \colon \ = \ (x = s) \vee \exists y \left[R(y) \wedge (E(y,x) \vee E(y,t))\right]$$

So, $\left[t \notin R^i \wedge \exists y \left(R^i(y) \wedge E(y,t)\right)\right] \Rightarrow \left[t \in R^{i+1} \wedge R^{i+1} = A\right] \Rightarrow \left[R^{i+1} = R^{i+2}\right]$. Since R only appears positively within ψ_4 the formula gives rise to a monotone operator, allowing:

$$\textsc{Min-SP} \equiv \mathit{depth}\left([\mathrm{LFP}_{R,x}\,\psi_4]\,(t)\right)$$

The examples given above show how to implement a problem from P_{opt} using both the IFP and LFP frameworks presented in Theorem 3.4.1 and Corollary 3.4.2 respectively. For both cases, the formulae given that define the operators are in first-order logic as they contain no nested IFP or LFP operators. If there is a strict syntactic hierarchy within the frameworks, then Corollary 3.4.3 demonstrates its ceiling and the examples above its floor.

3.5.2 Max-Flow

Given a network with a source and a sink, the maximisation problem Max-Flow asks: "what is the maximum achievable flow through the network from the source to the sink?" In this section the variant Max-Unit-Flow shall be considered, in which the network is represented as a (directed) graph and the weight of each edge is one. It is formally defined in Section 2.2.3.1.

Input structures for Max-Unit-Flow use the same vocabulary as those for Min-SP (see Section 3.5.1), which is $\sigma_{Gst} = \langle E^2, s, t\rangle$ with s being the network source, t being the network sink and E being the links between nodes in the network. The algorithm presented here for solving this problem follows on from that of Min-SP (see Chapter 1 of [Pap94]) and is based on the idea of *bottlenecks*. Effectively the maximum flow is equal to the maximum number of disjoint paths in the graph between s and t, and a bottleneck must exist on the shortest path between s and t. This leads to an algorithm that finds the shortest path between s and t and then *removes* it from the graph, before repeating the process. The number of times it finds and removes a path is equal to the maximum flow; when it cannot find a path the algorithm terminates.

Since this algorithm solves Max-Unit-Flow in $O(n^5)$ it is expected that a representation of it using fixed-point logic to either construct a relation of arity 5, or for there to be multiple nested fixed-point operators of a lower arity.

The algorithm used to solve the Min-SP problem in Section 3.5.1 did so by traversing the graph in a breadth-first manner from vertex s and counting the number of edges traversed when vertex t is encountered. What it does *not* do is construct the actual shortest path, which is what the algorithm for Max-Unit-Flow requires. To construct a shortest path, a breath-first traversal of the graph that marks the edges, as opposed to vertices, visited is required, along with a method for building just one such path.

Take the following formula:

$$\begin{aligned}\varphi_1(T,R,x,y) &= [E(x,y) \wedge \neg T(x,y)] \\ &\wedge\ [(x = s \wedge \forall u.\ (E(s,u) \Rightarrow y \leq u)) \vee \exists z\, R(z,x)]\end{aligned}$$

The inflationary fixed-point of φ_1, for some fixed relation T, will contain a tuple (x,y) if it is an edge, is not in T and is either the first edge lexicographically emanating from s or is connected to some edge already in R. This effectively produces a relation that

contains all the edges that lie on a path between s and t from the graph $G' = \langle E \setminus T, s, t \rangle$, where the paths all traverse the first (lexicographically) edge to emanate from s in G'.

Using the least-fixed point of φ_1, an actual path between s and t can be constructed. This is done by starting at t and traversing edges in $[\mathrm{IFP}_{R,x,y}\, \varphi_1(T)]$ in reserve, until s is reached. Given the following formula:

$$\begin{aligned} \varphi_2(T, S, x, y) &= [\mathrm{IFP}_{R,x,y}\, \varphi_1(T)]\,(x, y) \\ &\wedge \forall u\ ([\mathrm{IFP}_{R,x,y}\, \varphi_1(T)]\,(u, y) \Rightarrow x \leq u) \\ &\wedge [y = t \vee \exists z\, S(y, z)] \end{aligned}$$

Again, the inflationary fixed-point of φ_2 is calculated for some fixed T, which is also fixed when constructing $[\mathrm{IFP}_{R,x,y}\, \varphi_1(T)]$. The relation S^i contains a fragment of a path between s and t, in fact it contains the final i edges of a path, since it is building it in reverse. The inflationary fixed-point of φ_1 may contain edges from the graph that are incident on the same vertex; this happens when two paths collide. As such, when traversing a path in reverse a choice must be made as to which way to go, since if both edges were added to the path, then it would in fact not be a path. The logic fragment $\forall u\ ([\mathrm{IFP}_{R,x,y}\, \varphi_1(T)]\,(u, y) \Rightarrow x \leq u)$ ensures that the lexicographically first option is taken where two paths collide.

Up until now, the relation T and its purpose have not been defined. T is used to record the edges of the paths that have already been constructed in order to ensure that any new path is disjoint from these; the formula φ_1 only adds edges to the relation R if they are not in T. So at each iteration, a path is constructed by computing the inflationary fixed point of φ_2 with respect to T^i, and is then added to T^{i+1}. The maximum flow in the network equals the number of distinct paths which equals the inductive depth of the inflationary fixed-point of the following formula:

$$\begin{aligned} \varphi_3(T, x, y) &= (\exists u \exists v\ ([\mathrm{IFP}_{R,x,y}\, \varphi_2(T)]\,(s, u) \wedge [\mathrm{IFP}_{R,x,y}\, \varphi_2(T)]\,(v, t)) \\ &\quad \wedge [\mathrm{IFP}_{R,x,y}\, \varphi_2(T)]\,(x, y) \wedge T \neq \emptyset)) \\ &\quad \vee (x = s \wedge y = s) \end{aligned}$$

Which gives the optimal value as:

$$\mathrm{opt}_{MuF} = \mathit{depth}\,([\mathrm{IFP}_{T,x,y}\, \varphi_3]\,(s, s))$$

The formula φ_3 only adds the tuple (s, s) when $T = \emptyset$, i.e. at T^0. This is done to ensure that at least one iteration occurs so that the inductive depth of the inflationary fixed-point of φ_3 is at least zero and never $\perp$ (undefined); every network has a maximum flow. On subsequent iterations (T^i where $i \geq 1$), φ_3 ensures that there is actually a complete path between s and t and then adds every edge from that path to T^{i+1}. If no such path exists then no edges are added and $T^{i+1} = T^i$. The inductive depth of φ_3 is therefore equal to the number of paths found (the dummy clause added on the first iteration is there to offset the fact that the inductive depth is the number of iterations, minus one).

3.6 A Horn Logic Framework for P_{opt}^{PB}

Having successfully developed a framework for characterising P_{opt}^{PB} using Immerman's result that FO(IFP) = P, it is now prudent to consider why Manyem and Bueno's approach using Grädel's result that SO-Horn = P, failed. Firstly, let us reconsider what it means for a framework to characterise a class of problems:

Definition 3.6.1. In order that a framework $\mathcal{F}$ characterises a class of problems K, it must be shown that:

1. Any problem from K can be defined using the framework.
2. Any problem that the framework $\mathcal{F}$ can define is itself a problem in K.

In showing the second part of the above definition, it (usually) suffices to find a problem $\mathcal{Q}$ that can be used to compute the optimal value of any problem described using the framework and show that this problem (the so called *framework solver*) is itself in K.

The key undoing of Manyem and Bueno's frameworks is that it has been shown in Theorem's 3.3.4 and 3.3.6 that problems complete for classes further up the hierarchy (from NP_{opt}^{PB}) can be described using them. Another problem is that they treat maximisation and minimisation problems separately and symmetrically, but since P = co-P there is no need for this separate treatment of P_{max}^{PB} and P_{min}^{PB} since a problem in one can be reduced to a problem in the other in an amount of time at most polynomial in the input size. Finally, they use a tuple counting framework, which as discussed in Remark 3.2.8, which gives rise to some technical problems when dealing with minimisation problems.

With all this in mind, along with the key results in Sections 3.3 and 3.4, an avenue of research was pursued that sought to build a framework for P_{opt}^{PB} using Grädel's result. This framework uses the arity of a relation as its measure for the optimal value (as in [KT95]) and does not distinguish between maximisation and minimisation problems. Next is a definition of a family of frameworks similar to those from Theorem 3.2.7, but with the first-order component restricted to Horn formulae.

Definition 3.6.2. Let $\mathcal{Q} = (I_{\mathcal{Q}}, F_{\mathcal{Q}}, \text{cost}, \max)$ be a maximisation problem over σ and let $i \geq 1$. The problem $\mathcal{Q}$ is in MAX F Π_i-Horn if, and only if, there exists a signature $\tau = \langle S_1, S_2, \ldots, S_t \rangle$, consisting entirely of relation symbols and disjoint from σ, and a Π_i first-order sentence φ over $\sigma \cup \tau$ that is in Horn form with respect to τ (see Definition 2.1.65) such that for every instance $\mathcal{A}$ in $I_{\mathcal{Q}}$:

$$\text{opt}_{\mathcal{Q}}(\mathcal{A}) = \max_{\mathcal{B}} \{|S_1| : (\mathcal{A}, \mathcal{B}) \models \varphi\},$$

where $\mathcal{B}$ ranges over all τ-structures with domain A.

There are analogous definitions of MAX F Σ_i-Horn, MIN F Π_i-Horn, and MIN F Σ_i-Horn, similar to the classes defined in the postscript of Theorem 3.2.7.

The framework that is a candidate for characterising P_{opt}^{PB} is MIN F Π_1-Horn. The components required to construct a solver for this framework are introduced.

Lemma 3.6.3. *The decision problem* Horn-Sat *(c.f. Theorem 2.1.64) is in the complexity class* P.

Proof. Polynomial time algorithms based on the concept of *unit propagation* are well studied (see [CLRS09] and specifically Chapter 5 of [BL99]). A description of the problem (taken from Chapter 9 of [Imm99]) in the logic FO(LFP) over structures of the vocabulary σ_{SAT} (see Definition 2.1.58) is given by

$$\varphi(R, x) = \exists c \left(P(x, c) \wedge \forall y \left(N(y, c) \Rightarrow R(y)\right)\right)$$

Let $T = \mathrm{LFP}_{\varphi}$, and the sentence

$$\psi = \forall c \exists x \left(P(x, c) \wedge T(x) \vee N(x, c) \wedge \neg T(x)\right)$$

defines the problem Horn-Sat. Thus Horn-Sat $\in$ P. □

An algorithm for solving Horn-Sat based on the concept of *unit propagation* takes advantage of the following three different classes of Horn clauses

1. Those that contain a single positive literal, x, which must always be true in any satisfying assignment.

2. Constraint clauses that contain no positive literals, $(x_1 \wedge \ldots \wedge x_k) \Rightarrow$ false, where the variables on the left hand side of the implication cannot all be set to true in any satisfying assignment.

3. Implication clauses that are of the form $(x_1 \wedge \ldots \wedge x_k) \Rightarrow x_{k+1}$, such that if all the variables on the left hand side of the implication are true, then so is the variable on the right hand side.

Using these rules to build a satisfying set of literals for the problem Horn-Sat (c.f. the set T from Lemma 3.6.3) leads to the minimum number of clauses being satisfied. This is because in any satisfying assignment literals of clause type 1 must always be set to true and positive literals of clause type 3 are set to true only if all of the negative literals are true (this has to be repeated recursively). These literals must be set for any satisfying assignment, therefore if any clauses of type 2 are unsatisfied, then the formula is unsatisfiable.

By augmenting Horn-SAT with the additional requirement that no more than k literals are set to true from a specific subset the following problem is derived:

Definition 3.6.4. The decision problem k-Min-Literal-Subset-Horn-SAT has input instances that are the same as the decision problem Horn-SAT (see Definition 2.1.63) but with a constant k and a set S of literals ($S \subseteq \{x_1, \ldots x_\alpha\}$). It asks the question: is there an assignment to the boolean variables $x_1, \ldots, x_\alpha$ such that every clause is satisfied *and* the number of boolean variables that are a member of S and set to true is $\leq k$?

When deciding if a Horn-SAT instance is satisfiable using the algorithm based on the concept of *unit propagation*, the constructed set of satisfying literals is always the minimum number required to satisfy all the clauses in the Horn formula. This observation leads to the following lemma:

Lemma 3.6.5. *The decision problem* k-Min-Literal-Subset-Horn-SAT *is in* P.

Proof. Apply the unit propagation algorithm to obtain a set of literals L. Restrict this set by removing all literals that are not in the subset S and if the cardinality of this restricted set $|L \cap S|$ is $\leq k$ accept the input instance, otherwise reject. Due to the fact that L is a *minimal* set of literals for satisfying the input Horn formula and the fact that unit propagation algorithm is deterministic (in the sense that it never has to choose between 2 or more literals to make true), it follows that $L \cap S$ is the minimal set of literals that are required in a satisfying assignment and are in S.

Assume that L and S are such sets of literals. Now assume that there is another assignment L' such that $|L'| \leq |L|$ and hence that $|L' \cap S| < |L \cap S|$. It must be the case that there is a literal in S that is in L but not in L'; let x be such a literal. In order that x is in L it must be the case that x appears in one of the following types of clauses:

1. $\text{true} \Rightarrow x$
2. $(y_1 \wedge \ldots \wedge y_l) \Rightarrow x$

Now, if it is the first type, if it is true in L then it must also be true in L'. If it is the second type then it must be the case that the literals $y_1 \ldots y_l$ are also true in L. Since the algorithm is deterministic these literals must also be true in L', leading to the conclusion that x must be true in L', which contradicts the original assumption. Hence the cardinality of $|L \cap S|$ is the minimal number of literals from S that must be true in any satisfying assignment. □

Since the decision problem k-Min-Literal-Subset-Horn-SAT is in P, the optimisation variant Min-Literal-Subset-Horn-SAT is in P_{opt}^{PB} (by Definition 2.2.11). The optimal solution is bounded by the number of literals, which is linear in the size of the input, hence the optimisation problem is polynomially-bounded.

Now the main theorem, which says that the framework $\mathsf{MIN}\ \mathsf{F}\ \Pi_1$-Horn characterises the class of polynomially-bounded P-optimisation problems is presented, using the problem k-Min-Literal-Subset-Horn-SAT as the framework solver.

Theorem 3.6.6. *A polynomially bounded* P*-minimisation problem* $\mathcal{Q} = (I_\mathcal{Q}, F_\mathcal{Q}, \text{cost}, \min)$*, is in the class* P_{opt}^{PB} *iff there exists an input vocabulary* σ*, a solution vocabulary* τ *and a quantifier-free first-order formula* η *that is appropriate to* $(\sigma \cup \tau)$ *and Horn with respect to* τ*, such that the optimal solution to* $\mathcal{Q}$ *is given by*

$$\text{opt}_\mathcal{Q}(\mathcal{A}) = \min_\mathcal{B}\{|S_1| \mid (\mathcal{A}, \mathcal{B}) \models \forall \bar{x}\, \eta\} \tag{3.14}$$

where $\mathcal{A} \in I_\mathcal{Q}$ *is an ordered structure,* A *is the universe of* $\mathcal{A}$*;* $\mathcal{B}$ *is a structure of vocabulary* τ *whose universe is also* A *and* S_1 *is some particular* k*-ary relation symbol from the solution vocabulary* τ*.*

Proof. Any problem that can be represented using the above framework must be polynomially bounded, since there are only $|A|^k$ different tuples that can be members of S_1 in any particular structure $\mathcal{B}$ and thus its cardinality is bounded by $|A|^k$.

Now suppose that $\mathcal{Q}$ is a polynomially bounded P-minimisation problem where the $\text{opt}_\mathcal{Q}(\mathcal{A}) \leq |A|^k$ for all structures $\mathcal{A} \in I_\mathcal{Q}$ and some fixed k. Let W be a new k-ary

relation and add it to the input structure. The class

$$K := \{(\mathcal{A}, W) \mid \mathcal{A} \in I_{\mathcal{Q}}, W \subseteq A^k, \text{there is an } \mathcal{S} \in F_{\mathcal{Q}} \text{ s.t. } \text{cost}(\mathcal{A}, \mathcal{S}) \leq |W|\}$$

is equivalent to the class of $(\mathcal{A}, c)$ structures that the decision variant of $\mathcal{Q}$ accepts on, with $c = |W|$. Thus the class K is in PTIME.

Since P = ∃SO-Horn (see Theorem 2.1.76), there exists some sentence $\exists \bar{R} \forall \bar{x}\, \eta(W, \bar{R}, \bar{x})$ that characterises the class K such that

$$(\mathcal{A}, W) \in K \Leftrightarrow (\mathcal{A}, W) \models \exists \bar{R} \forall \bar{x}\, \eta(W, \bar{R}, \bar{x})$$

The optimal solution to $\mathcal{Q}$ is any $(\mathcal{A}, W) \in K$ structure with the smallest W relation, thus:

$$\text{opt}_{\mathcal{Q}} = \min_{(W^{\mathcal{B}}, \bar{R}^{\mathcal{B}})} \{|W| \mid (\mathcal{A}, W^{\mathcal{B}}, \bar{R}^{\mathcal{B}}) \models \forall \bar{x}\, \eta\}$$

Let $\tau = (W, \bar{R})$ with $S_1 = W$, hence any such problem $\mathcal{Q}$ is definable in the framework given in Equation 3.14.

For the converse, it is required to show that for any formula written using the framework, the problem of finding the optimal value is in itself a polynomially-bounded P-optimisation problem. To do this, it suffices to show that the decision variant of the problem of solving the framework is in P.

Since $\forall x\, \varphi \equiv \varphi[x/x_1] \wedge \varphi[x/x_2] \wedge \ldots \wedge \varphi[x/x_n]$ the formula $\forall \bar{x}\, \eta$ can be expanded to a logically equivalent quantifier-free formula ψ, whose length is n^k times that of η (where $\bar{x}$ is a k-tuple). Note that if η is Horn with respect to τ, then the expanded formula ψ is still Horn with respect to τ , since Horn formulae (with respect to τ) are closed under conjunction. Following this expansion, the only unknowns in the formula ψ are the values of the relations and constants from the vocabularies σ and τ (note that τ is a *relational* vocabulary).

Fix the σ-structure $\mathcal{A}$ and substitute the values for the relation and constant symbols in σ into η. Following this the only unknowns are the values of the relation symbols from τ; everything else about the formula is known. Let $\psi_{\mathcal{A}}$ be the result of the substitution of values for σ from $\mathcal{A}$ in the expansion of η.

Without loss of generality, assume that the vocabulary $\tau = \langle S_1, S_2, ..., S_r \rangle$, where $r \geq 1$. From the formula $\psi_{\mathcal{A}}$ construct an input instance to the problem k-Min-Literal-Subset-Horn-SAT (see Definition 3.6.4) where each literal corresponds to a tuple being a member of a relation symbol from τ in the structure $\mathcal{B}$ and S is the subset of literals corresponding to a tuple being a member of the relation S_1. Call this mapping $\Phi(\mathcal{A})$ and let the target optimal value be m.

Each relation symbol S_i in τ gives rise to at most $n^{\alpha(i)}$ literals (recall that α is a function mapping relation symbols to arities). This means that there are at most $r.n^{max_\alpha}$ literals in $\Phi(\mathcal{A})$, where max_α is the largest arity of a relation symbol in τ, and so the size of $\Phi(\mathcal{A})$ is the function of some polynomial of the size of the universe of $\mathcal{A}$.

Now it remains to show that under the mapping Φ, the problem k-Min-Literal-Subset-Horn-SAT solves the MIN F Π_1-Horn framework, i.e. $\text{opt}_{\mathcal{Q}} \leq m$ iff k-Min-Literal-Subset-Horn-SAT accepts under the mapping Φ.

If k-Min Literal Subset Horn SAT accepts, then there are at most m tuples that are in the set S. Place all these tuples into the realisation of S_1 in $\mathcal{B}$, so $|S_1^{\mathcal{B}}| \leq m$ and thus $\text{opt}_{\mathcal{Q}} \leq m$.

Conversely suppose that k-Min-Literal-Subset-Horn-SAT rejects but that there exists some τ-structure $\mathcal{B}$ such that $|S_1| \leq m$. Since the problem rejects on the instance $\Phi(\mathcal{A})$ this means that there is no assignment to the literals in $\Phi(\mathcal{A})$ such that at most m literals in S are true. As under the mapping each literal in S corresponds to the membership of a tuple in S_1 and the mapping ensures that every possible tuple has a corresponding literal, it cannot be the case that there is a τ-structure $\mathcal{B}$ s.t. $|S_1| \leq m$, contradicting the original assumption.

Since, by Lemma 3.6.5, k-Min-Literal-Subset-Horn-SAT is in P it follows that whether $\text{opt}_{\mathcal{Q}} \leq m$ holds or not can be computed in polynomial-time, giving the required result. □

A direct consequence of the use of k-Min Literal Subset Horn SAT to prove the above theorem is

Corollary 3.6.7. *The problem* k-Min Literal Subset Horn SAT *is complete for all minimisation problems in the class* P_{opt}^{PB}.

3.7 Characterising Unbounded Optimisation Problems

So far, all the frameworks that have been presented for characterising optimisation problems have only dealt with polynomially-bounded problems. In this section, the characterisation of classes of problems that do not admit this restriction is considered.

3.7.1 Removing the Polynomially Bounded Restriction From the NP_{opt}^{PB} Frameworks

The frameworks that Kolaitis and Thakur give for characterising NP_{opt} (see Theorems 3.2.4 and 3.2.7) do so only for the polynomially-bounded subclass NP_{opt}^{PB} (see Definition 2.2.8 for exactly what is meant by *polynomially-bounded*). The framework proposed by Manyem and Bueno (see Theorem 3.2.10) also only dealt with the polynomially-bounded subclass P_{opt}^{PB}, as did the frameworks introduced earlier in this chapter in Theorems 3.4.1 and 3.6.6.

The reason for this is that all these frameworks model the objective function from an optimisation problem as a measure of either the number of tuples that satisfy some formula for a given solution (in general these are derived from [PY91, KT94]), or as the cardinality of a relation that is part of the solution structure (in general these are derived from [KT95]). Since for a k-ary relation over universe A there are only $|A|^k$ different tuples that can be its members, it can be seen that any objective function based on counting these members will always be limited to values bounded by the polynomial n^k.

Papadimitriou and Yannakakis conjectured that an exponentially bounded objective function could be modelled within their framework by assigning values, or weights, to each tuple $\bar{x}$, but they did not discuss this any further [PY91].

A generalisation of this idea of adding weights to each tuple is first given by Zimand, who extends Kolaitis and Thakur's new framework (from Theorem 3.2.7) to characterise NP_{opt} [Zim98]. He achieves this by introducing a weight function w, which maps some relation to a natural number. By using this function to model the values of the objective function, rather than just measuring the cardinality of the relation, it is then shown that arbitrary objective function values can be calculated. The framework is given by:

$$\mathrm{opt}_{\mathcal{Q}}(\mathcal{A}) = \underset{\mathcal{B}}{\mu}\{w(S_1) \mid (\mathcal{A}, \mathcal{B}) \models \varphi\} \tag{3.15}$$

Where $\mathcal{A}$ is a σ-structure (the input structure); $\mathcal{B}$ is a τ-structure, with S_1 some particular k-ary relation in τ; φ is a first-order formula appropriate to (σ, τ) and w is a function that maps from sets of k-tuples to natural numbers.

The similarities between the above framework and the new polynomially-bounded NP-optimisation framework presented in Equation 3.8 of Theorem 3.2.7 are obvious; the function w has simply replaced the cardinality operator.

3.7.1.1 Representing Exponentially Bounded Values in Structures

It is fair to question not just how a logical framework for optimisation problems can extract an unbounded value, but also how these values are represented within a relational structure in the first place. Recall that the preferred model of computation, Turing machines, operates on strings, which can be encoded as binary strings (see Section 2.1.2 for more details). Now, a positive integer m can be encoded as a binary string of length $\lceil \log_2(m+1) \rceil$; or more so, a binary string of length l can be used to encode all positive integers less than 2^l. Note that it is common to refer to a binary string of length l as being a string of *l-bits*.

With this in mind the vocabulary of weighted graphs, where the weights are arbitrary positive integers, is introduced:

Definition 3.7.1. The vocabulary of weighted (di-)graphs is an extension of that of unweighted (di-)graphs (see Definition 2.1.5) to include the weight function w that maps from a pair of vertices to a weight. More formally $\sigma_{Gw} = \langle E^2, w^2 \rangle$ with w being a 2-ary function and having the following properties: $w : V^2 \mapsto \mathbb{N}$, where V is the set of vertices/universe of the structure and for all edges (u, v) it holds that $w(u, v) \geq 0$ iff $E(u, v)$.

Since the structures being dealt with in this Thesis have all so far been relational, it is now prudent to show how to build a weighted graph using a purely relational structure. In order to do this a method of encoding weights using relations is needed. One such way to do this is based on the observation that whilst a k-ary relation over universe A can only have at most $|A^k|$ members (which is equal to $|A|^k$ and hence polynomial in the size of the input) there are $2^{|A|^k}$ *combinations* of tuples that it could contain; this is because the cardinality of the power set $|\mathcal{P}(A^k)| = 2^{|A|^k}$. Now, by assigning each of these combinations to a number, it is possible to encode the weights using a relation, where the relation is in effect representing a $|A|^k$-bit number.

3.7.1.2 Zimand's Theorem

The principle that Zimand uses to model an arbitrary objective function is based on the observation of the above stated fact: that while there are only polynomially many *unique* tuples that can be members of a relation, there are exponentially many *combinations* of these tuples that the relation can be assigned to. This leads him to propose augmenting Kolaitis and Thakur's new framework from Theorem 3.2.7 with a weight function w, as stated in Equation 3.15. In order for such a function w to give an exponentially bounded range of objective values, it must provide a one-to-one mapping between sets of tuples and natural numbers, as captured by the following definition.

Definition 3.7.2. The weight function w has the following properties:

1. $w : A^k \mapsto \mathbb{N}$, where k is the arity of the relation S_0 in equation 3.15.
2. $w(\{\bar{x}_i\}) = w(\{\bar{x}_j\}) \Leftrightarrow \bar{x}_i = \bar{x}_j$.
3. $w(S_i^{\mathcal{B}_1}) = w(S_j^{\mathcal{B}_2}) \Leftrightarrow S_i^{\mathcal{B}_1} = S_j^{\mathcal{B}_2}$.
4. $w(S_i) = \sum_{\bar{x} \in S_i} w(\bar{x})$.
5. The range of values of w is from zero up to $\left(2^{\|A\|^k+1} - 1\right)$ inclusive.

Effectively, w maps between the binary encoding of a number, where each bit is represented by the presence of a tuple in a k-ary relation, to the number itself. In order to construct this function, a total ordering over the tuples in A^k is needed. With this, it is possible to give each tuple a value corresponding to its position in the ordering. If some tuple $\bar{x} \in A^k$ is in position i in the ordering, that is, it is the i^{th} tuple (in the range 1 to $\|A\|^k$ inclusive), then $w(\{\bar{x}\}) = 2^{(i-1)}$.

In order for the weight function w to be a valid model of an exponentially bounded objective function, it must be able to be both constructed and queried in polynomial time (see Definition 2.2.2).

Lemma 3.7.3. *The weight function w (see Definition 3.7.2), whose domain is A^k, can be both constructed and queried in polynomial time.*

Proof. Assume that the universe $A = \{1, 2, \ldots, n\}$ and that it is ordered according to the natural order of its elements. Using this ordering, what is needed is to write out the tuples from A^k in order. What follows is a method for constructing w by induction on k.

To write out the tuples from $A^{(k+1)}$ in order, write out all the tuples from A^k in order, but augment them with an element from the universe A. Let t_i^k represent all the elements from the i^{th} tuple from A^k, then start by writing out the augmented tuples $(t_i^k, 1)$, (where i counts through the range $1 \leq i \leq |A|^k$), followed by $(t_i^k, 2)$ and so on up to (t_i^k, n). This constructs an ordering for the tuples from $A^{(k+1)}$ assuming that an ordering for the tuples from A^k is already at hand. For the base case (where $k = 1$), the weight function is simply constructed by writing out the elements of the universe A in order. Hence the function w can be constructed in this way using $|A|^k$ time, which is polynomial in n.

This list can then be used to calculate the value of some relation R of arity k. Start by assigning an area for writing down the output value, and initialise it with $|A|^k$ zeros.

For each tuple in R, search through the list until a match is found. If the match is in the i^{th} position then write a 1 in the i^{th} position of the output value area. Once this has been done for all tuples in R, a binary representation of the number $w(R)$ is written in the output value area. Each tuple takes at most $|A|^k$ steps to find (as this is the length of the constructed function), and this must be repeated at most $|A|^k$ times, as this is the maximum number of tuples in R. Hence the upper bound is $O(n^{2k})$ and the function w can be queried in polynomial time, if constructed as above. □

It is noteworthy that in order to construct the weight function w it is required that the universe of the structure is *ordered*. Often, the input structure to a non-polynomially-bounded NP-optimisation problem will already contain an ordering because it encodes exponentially large numbers as l-bit strings, which requires knowledge of an ordering on the universe in order to be meaningful. In these cases, this ordering can be used to construct the weight function w.

There are problems that do not necessarily contain an ordering in the input structure. One such example is that trivial[1] optimisation problem that given a polynomially-bounded number x (such that $x \leq n^k$ for some fixed k), it asks the question: what is the greatest value of 2^x? Clearly the objective function for this problem is required to encode x-bit numbers and so requires an ordering on the universe.

Due to the existence of these problems, it is required, within Zimand's framework, for the input structures to either be ordered or contain a suitable weight function already.

The main theorem from his work is now presented.

Theorem 3.7.4 (Zimand's Theorem [Zim98])**.** *Let $\mathcal{Q}$ be an (unbounded)* NP*-optimisation problem whose input instance $\mathcal{A}$ is an ordered structure of vocabulary σ. For some fixed k there exists a k-ary weight function w, a vocabulary τ and a first-order Π_2 formula φ appropriate to $\sigma \cup \tau$ such that the optimal value of $\mathcal{Q}$ is given by:*

$$\text{opt}_{\mathcal{Q}}(\mathcal{A}) = \underset{\mathcal{B}}{\mu}\{w(S_0) \mid (\mathcal{A}, \mathcal{B}) \models \varphi\}$$

where $\mathcal{B}$ ranges over all τ-structures, $\mu \in \{\max, \min\}$ and S_0 is some particular k-ary relation from τ.

Proof. The proof that an NP-optimisation problem $\mathcal{Q}$ can be represented in the framework proceeds by introducing a new relation R of arity k and defining the following class of problems when $\mu = \max$:

$$K_1 := \{(\mathcal{A}, R) \mid \mathcal{A} \in I_{\mathcal{Q}}, R \subseteq A^k, \text{ there is a } \mathcal{B} \in F_{\mathcal{Q}}(\mathcal{A}) \text{s.t. } \text{cost}(\mathcal{A}, \mathcal{B}) \geq w(R)\}$$

and when $\mu = \min$:

$$K_1 := \{(\mathcal{A}, R) \mid \mathcal{A} \in I_{\mathcal{Q}}, R \subseteq A^k, \text{ there is a } \mathcal{B} \in F_{\mathcal{Q}}(\mathcal{A}) \text{s.t. } \text{cost}(\mathcal{A}, \mathcal{B}) \leq w(R)\}$$

[1] Trivial in the sense that there is only one feasible solution and so any feasible solution is automatically the optimal.

which corresponds to the decision variant $\mathcal{Q}_D$. Since $\mathcal{Q}_D$ is in NP (by the Definition of an NP-optimisation problem in 2.2.2), there is a formula $\exists \bar{S}\, \psi$, where ψ is a first-order Π_2 formula, that characterises the class K_1 (and hence $\mathcal{Q}_D$). That is $(\mathcal{A}, R) \in K_1 \Leftrightarrow (\mathcal{A}, R) \models \exists \bar{S}\, \psi$, noting that ψ is appropriate to the vocabulary $\sigma \cup \langle R^k \rangle$. With this, it can be seen that finding the optimal solution to the problem $\mathcal{Q}$ is the same as finding some extended structure $(\mathcal{A}, R) \in K_1$ with the property that $\text{opt}_{\mathcal{Q}}(\mathcal{A}) = w(R)$, thus:

$$\text{opt}_{\mathcal{Q}}(\mathcal{A}) = \underset{(\mathcal{C},R)}{\mu} \{w(R) \mid (\mathcal{A}, \mathcal{C}, R) \models \psi\}$$

which is in the form required by this framework.

Conversely, given an instance of the framework, the following NP-optimisation problem can be constructed: $I_{\mathcal{Q}}$ is the set of all σ-structures; $F_{\mathcal{Q}}(\mathcal{A})$ is set of all τ-structures ($\mathcal{B}$); and the function $\text{cost}(\mathcal{A}, \mathcal{B})$ is defined as the value of $w(S_0^{\mathcal{B}})$, which is computable in polynomial time due to Lemma 3.7.3. Since all optimisation problems defined in this way are NP-optimisation problems (see Remark 2.2.3), it is unnecessary to show that the decision variant of this constructed problem is in NP, although it clearly is: guess a τ-structure $\mathcal{B}$ and check that $w(S_0^{\mathcal{B}}) \geq k$ if $\mu = \max$, or $w(S_0^{\mathcal{B}}) \leq k$ if $\mu = \min$, where k is the target value. □

There are many practical barriers to representing a problem using the framework proposed by Zimand (see equation 3.15) for capturing NP-optimisation problems that are not polynomially bounded [Zim98]; it appears that no examples of applications of the framework to real problems have been published.

Chapter 4

Modal Logic, Hybrid Graph Logic and Games

Classical modal and temporal logics were developed to argue about any system in which there are relationships or transitions between states. The ubiquitous mathematical structure for representing such discrete systems is the graph and so it comes as no surprise that a directed graph is used to represent the underlying *modal frame* in these logics (see Definition 2.3.1). In computer science, modal logics are often used in the field of Formal Methods. In this field a system is modelled as a class of relational structures and a formula is then written in an appropriate logic so that it describes some property of the system. To verify that the system is correct this property is then checked to see if it is either: always true of the system (valid), never true of the system (unsatisfiable), sometimes true of the system (satisfiable) or sometimes false of the system (invalid). These four questions are the same as the *satisfiability* and *validity* problems of the logic in question (plus their complements). If the logic used was first-order logic then the verification of the system would not be possible, since both satisfiability and validity are undecidable for first-order logic. Since these problems are decidable for basic modal logic and its extensions these are commonly used in Formal Methods so that the verification of the system is (at least) possible.

4.1 Chapter Outline

Firstly, this chapter sets the scene with past research on hybrid modal logic and in particular Hybrid Graph Logic (4.2); it then demonstrates some graph problems definable using Hybrid Graph Logic (4.3); next it introduces games that can be used to demonstrate whether two structures are logically equivalent or not and describes how these can be used to show whether graph properties are definable in a class of Hybrid Graph Logic or not (4.4); the games are then played on graph structures and strict hierarchies of classes of formulae are derived (4.5).

4.2 Past Research

Hybrid logics have recently been developed and studied. A hybrid logic is an extension of a modal (or temporal) logic in which symbols are used to name individual points in a modal frame. Key to many hybrid logics is the use of *nominals*, n, and their corresponding nominal operators, $@_n$, which allow the current state to 'jump' to the point of a structure named by the nominal n. Another operator introduced by hybrid logics is the 'binder' $\downarrow$, which allows so called *world variables* to be bound to the current state. The study of hybrid logics in theoretical computer science has primarily been with regard to their use in model checking and verification of Formal Methods; consequently the focus has been on the decidability and complexity of the related model-checking, satisfiability and validity problems (see, for example, [AtC07, CS01, FdR06, MMS$^+$10]).

Up until recently, hybrid logics have not been closely studied within the context of Finite Model Theory and Descriptive Complexity. However, in [BS09] Benevides and Schechter defined (amongst other hybrid logics) Hybrid Graph Logic (HGL), which is basic modal logic (see definitions from Section 2.3.1) augmented with the use of nominals and a facility to verify the existence of paths in graphs (see HGL specific definitions from Section 2.3.2). The intention in [BS09] was to develop (modal and hybrid) logics for reasoning about graphs (as opposed to models) that are expressive enough to define core graph-theoretic problems relating to properties such as connectedness, acyclicity and Hamiltonicity. Actually, Hybrid Graph Logic *without* nominals is a well-known and well-studied fragment of both PDL and CTL (see, for example, [EH85]), and the logic itself has been independently formulated and studied (from the perspective of tableaux systems) in [KS10] where it is referred to as basic modal logic extended with nominals and eventualities. Here the logic is referred to as HGL, given that the emphasis of the research follows the tone set in [BS09].

In relation to expressibility, Areces, Blackburn and Marx [ABM01] studied the hybrid logic $\mathcal{H}(\downarrow, @)$, which is the extension of basic modal logic with the binder $\downarrow$ and nominal operators of the form $@_n$. They characterised its expressibility (on models) in terms of a fragment of first-order logic (namely the fragment that is invariant under generated sub-models) before developing Ehrenfeucht-Fraïssé style games and notions of bisimulation in doing so. Their investigation is primarily with regards to models although they briefly mention the relevance of their work to modal frames. They mention the study of the hybrid logic $\mathcal{H}(@)$, in relation to its expressibility, as being an interesting direction for further research. Note that HGL is more expressive than $\mathcal{H}(@)$ as it includes an extra modality corresponding to the non-reflexive transitive closure of the edge relation in the modal frame; within this naming convention HGL is more accurately termed $\mathcal{H}(@, +)$.

The focus of this chapter is on using Hybrid Graph Logic as a logic for defining problems involving finite directed graphs, which are effectively the same as finite modal frames. The logic itself is basic enough that it forms a fragment of many other hybrid logics yet within it computationally hard problems can be defined (assuming that $\mathsf{P} \neq \mathsf{NP}$); hence, it makes sense to undertake a fundamental study of this logic. As was noted in [BS09], results from [ABM00] and [FdR06], respectively, show that the satisfiability and validity problems for HGL are $\mathsf{EXPTIME}$-complete and the model-checking problem for HGL is solvable in time that is linear in the size of the model and the length of the formula.

4.3 Some problems definable in HGL

In this section some problems definable in Hybrid Graph Logic are exhibited; two of these problems featured strongly in [BS09].

4.3.1 Connectivity

The problem Connectivity consists of those digraphs for which there is a directed path from any vertex to any other vertex; that is, those digraphs that are strongly connected. This problem can be defined by the following formula φ of Hybrid Graph Logic:

$$\neg n \Rightarrow \Diamond^+ n$$

where n is a nominal.

To see this, suppose that the digraph $\mathcal{G} = \langle V, E\rangle$ is strongly connected but $\mathcal{G} \not\models \varphi$ (in particular, $\mathcal{G}$ has at least 2 vertices). There is a valuation function $\mu : \{n\} \to \mathcal{P}(V)$, with $\mu(n) = \{v\}$, and a point u of $\mathcal{G}$ such that $(\mathcal{G}, \mu, u) \not\models \neg n \Rightarrow \Diamond^+ n$. So, $u \neq v$ and for every vertex v' of the digraph $\mathcal{G}$ reachable from u via a non-trivial path (that is, a path with at least one edge), it must be the case that $v' \neq v$. However, every vertex of $\mathcal{G}$ different from u (of which v is one) is reachable from u via a non-trivial path and so this yields a contradiction. Conversely, suppose that $\mathcal{G} \models \varphi$ but that the digraph $\mathcal{G}$ is not strongly connected. So, there exist distinct vertices u and v for which there is no path from u to v in the digraph $\mathcal{G}$. Define $\mu(n) = \{v\}$. In particular, $(\mathcal{G}, \mu, u) \models \neg n \Rightarrow \Diamond^+ n$. So, there is a path in the digraph $\mathcal{G}$ from u to v, which yields a contradiction.

This places the problem Connectivity in the class $\mathrm{HGL}_1(0, 1)$.

4.3.2 Acyclic

The problem Acyclic consists of those digraphs that do not have a directed cycle containing at least one edge as a sub-digraph (note that self-loops are regarded as cycles of length 1). This problem can be defined by the following formula φ of Hybrid Graph Logic:

$$@_n \neg \Diamond^+ n,$$

where n is a nominal.

To see this, suppose that the digraph $\mathcal{G} = \langle V, E\rangle$ is acyclic but $\mathcal{G} \not\models @_n \neg \Diamond^+ n$. So, there exists a valuation function $\mu : \{n\} \to \mathcal{P}(V)$ and a point v of $\mathcal{G}$ such that $(\mathcal{G}, \mu, v) \not\models @_n \neg \Diamond^+ n$; that is, such that $(\mathcal{G}, \mu, v') \not\models \neg \Diamond^+ n$, where $\mu(n) = v'$, which is equivalent to $(\mathcal{G}, \mu, v') \models \Diamond^+ n$. Hence, there is a non-trivial path from v' to v' in the digraph $\mathcal{G}$ and a contradiction is obtained. Conversely, suppose that $\mathcal{G} \models @_n \neg \Diamond^+ n$ but the digraph $\mathcal{G}$ contains a cycle. Let v be a vertex on a cycle in $\mathcal{G}$ and define $\mu : \{n\} \to \mathcal{P}(V)$ via $\mu(n) = \{v\}$. This results in $(\mathcal{G}, \mu, v) \models @_n \neg \Diamond^+ n$; that is, $(\mathcal{G}, \mu, v) \models \neg \Diamond^+ n$, which yields a contradiction.

This places the problem Acyclic in the class $\mathrm{HGL}_2(0, 1)$.

4.3.3 Non-4-Colour-SC

The problem Non-4-Colour-SC consists of those digraphs that are strongly connected and can not be properly 4-coloured; the existence of a directed edge (u, v) in a digraph means that u and v must be coloured differently in any proper colouring. This problem can be defined by the following formula φ of Hybrid Graph Logic:

$$\begin{aligned}(\neg n \Rightarrow \Diamond^{+} n) \\ \wedge \Diamond^{+}((p_1 \wedge p_2 \wedge \Diamond(p_1 \wedge p_2)) \vee (p_1 \wedge \neg p_2 \wedge \Diamond(p_1 \wedge \neg p_2)) \\ \vee(\neg p_1 \wedge p_2 \wedge \Diamond(\neg p_1 \wedge p_2)) \vee (\neg p_1 \wedge \neg p_2 \wedge \Diamond(\neg p_1 \wedge \neg p_2)))\end{aligned}$$

where n is a nominal and p_1 and p_2 are both propositional symbols.

To see this, let $\mathcal{G} = \langle V, E\rangle$ and let $\mu : \{p_1, p_2\} \cup \{n\} \to \mathcal{P}(V)$ be any valuation function. μ is interpreted as providing a 4-colouring of the vertices of V with the colours being: $v \in \mu(p_1)$ and $v \in \mu(p_2)$; $v \in \mu(p_1)$ and $v \notin \mu(p_2)$; $v \notin \mu(p_1)$ and $v \in \mu(p_2)$; and $v \notin \mu(p_1)$ and $v \notin \mu(p_2)$. Suppose that $\mathcal{G} \models \varphi$. So, $\mathcal{G}$ is strongly connected and for every valuation function $\mu : \{p_1, p_2\} \cup \{n\} \to \mathcal{P}(V)$ (that is, every 4-colouring of the vertices of V) and for every $u \in V$, there exists a vertex v reachable via a non-trivial path from u so that there is an edge $(v, v') \in E$ where v and v' have the same colour. Hence, $\mathcal{G}$ is strongly connected and can not be properly 4-coloured. Conversely, if $\mathcal{G}$ is strongly connected and can not be properly 4-coloured then $\mathcal{G} \models \neg n \Rightarrow \Diamond^{+} n$ and no matter which valuation function $\mu : \{p_1, p_2\} \cup \{n\} \to \mathcal{P}(V)$ and point u of $\mathcal{G}$ is chosen, there is a non-trivial path from u to a vertex v from which there is an edge (v, v') with v and v' identically coloured; that is, $\mathcal{G} \models \varphi$.

This places the problem Non-4-Colour-SC $\in \mathrm{HGL}_2(2, 1)$.

4.3.4 Complexity and Decidability of Hybrid Graph Logic

The example in Section 4.3.3 above, illustrates that Hybrid Graph Logic can be used to define problems that are probably not solvable in polynomial-time, as Non-4-Colour-SC is co-NP-complete, where co-NP is the complement (see Definition 2.1.47) of NP (see Definition 2.1.41). To see this, observe that the graph problem K-Colour (also know as the graph Chromatic-Number problem) is NP-complete for all fixed $K \geq 3$ (see problem GT4 in Appendix A1 of [GJ79]) and so the problem 4-Colour is NP-complete. The addition of the *strongly connected* requirement doesn't make the problem any easier, hence 4-Colour-SC is also NP-complete and its complement is co-NP-complete. This example also shows that Hybrid Graph Logic cannot characterise, using stable formulae, an NP-complete problem (see Definition 2.1.50) unless NP = co-NP (i.e. NP is *closed under complement*).

It is true that hybrid modal logic with universal access has a decidable satisfiability (see Definition 2.3.6) problem [ABM00, FdR06] and that by Lemma 2.3.8 the validity problem (see Definition 2.3.7) is also decidable. Now, the key difference between hybrid modal logic with universal access and Hybrid Graph Logic is that while validity is the same for both logics, satisfiability in the later is the same as looking, in the former, for a satisfiable frame that is also valid. This means that validity is decidable in HGL, but

that Lemma 2.3.8 cannot be applied to HGL's validity problem to obtain an algorithm for solving satisfiability. In fact, the global satisfiability problem, (that is given a formula does there exist a model that globally satisfies φ?), is undecidable for basic modal logic (see, e.g. Chapter 6, Section 5 of [BdRV01]), so it follows that frame satisfaction is also undecidable for basic modal logic, and hence it is undecidable for HGL.

4.3.5 Research Question

The formulation of formulae defining problems such as Connectivity, Acyclicity and Non-4-Colour-SC has lead to the posing of the main question studied by the research in this chapter:

> What is the relative expressivity of fragments of Hybrid Graph Logic, obtained by restricting: the application of the operators $\Diamond$, $\Diamond^+$, $\Box$, $\Box^+$ and @; the number of propositional symbols used; and the number of nominals used?

As mentioned earlier, HGL is obtained from a well-studied fragment of PDL and CTL by allowing the use of nominals, and so it makes sense to better understand its expressibility, given that it will be a fragment of many well-known logics that are extended by the incorporation of nominals. Also, the study of the expressivity of "low-level" logics in descriptive complexity is common-place; see, for example, the study of finite variable or quantifier-prefix fragments of first-order logic [EF99, Lib04]. The logic HGL forms such a "low-level" logic and is readily extendible to obtain new logics, the expressibility of which will be built upon the expressibility of HGL.

While HGL does not have a decidable satisfiability problem it is arguable whether this would be useful in a formal verification environment; asking whether there exists some graph that has a certain property appears to be a counter-intuitive question, as it is already know what property of graphs the formula represents and so the construction of a satisfying graph should be trivial. It is much more likely that the graph will be fixed and that the question will be whether a property holds for all possible valuation functions and vertices of this fixed graph, which is decidable.

It does on the other hand have a decidable validity problem, allowing the model checking of properties expressed in HGL(P, N) to see if they are true for all directed graphs.

4.4 Games and Hybrid Graph Logic

One approach to separating fragments of Hybrid Graph Logic (or any logic for that matter) is to exhibit a problem (i.e. a class of graphs K) that can be characterised in one fragment, say $\mathcal{L}_1$ but not the other, say $\mathcal{L}_2$. This is done by taking two structures $\mathcal{A}$ and $\mathcal{B}$, one that is in the class of graphs, say $\mathcal{A} \in K$ and one that isn't, say $\mathcal{B} \notin K$ and demonstrating the structures are $\mathcal{L}_1$-equivalent but not $\mathcal{L}_2$-equivalent. This method is presented in Propositions 2.1.81 and 2.1.82, with logical/structural equivalence defined in Section 2.1.1.2 and Definition 2.3.21 giving the structural equivalence relations for fragments of HGL.

Showing that $\mathcal{A} \not\equiv_{\mathcal{L}_2} \mathcal{B}$ is straightforward: simply show that there is a formula in $\mathcal{L}_2$ that holds for $\mathcal{A}$ but not $\mathcal{B}$. The difficultly comes in showing that $\mathcal{A} \equiv_{\mathcal{L}_1} \mathcal{B}$, as

every possible formula within the fragment $\mathcal{L}_1$ must be considered. This is where the Ehrenfeucht-Fraïssé style pebble games and their modal logic versions for bisimulation come in.

4.4.1 Games on Pointed Structures (Models)

The game presented in this section aims to equate logical equivalence of two pointed structures with a winning strategy for one of the players (the Duplicator). It is derived from the game theoretic characterisation of bisimulation (see Section 2.3.3 and Definition 2.3.24), which is in turn derived from the Ehrenfeucht-Fraïssé game for first-order logic (see Section 2.1.5.2).

Definition 4.4.1. Let $r \geq 0$, let P be a set of propositional symbols and let N be a set of nominals. The *r-round* $\mathrm{HGL}(\mathsf{P},\mathsf{N})$*-game* is a two-player pebble game played by Spoiler and Duplicator. It is played on two pointed $\mathsf{P} \cup \mathsf{N}$-structures $(\mathcal{G}, \mu, u)$ and $(\mathcal{H}, \lambda, v)$. Initially, prior to any play of the game, pebble a sits on point u of $\mathcal{G}$ and pebble b sits on point v of $\mathcal{H}$. During a play of the game, pebble a is moved from point to point in $\mathcal{G}$, with pebble b being moved from point to point in $\mathcal{H}$. There are r rounds in any play of the game and the point of $\mathcal{G}$ (resp. $\mathcal{H}$) on which pebble a (resp. b) sits after round i of some play is denoted by a_i (resp. b_i); so, $a_0 = u$ and $b_0 = v$. Each round i of a play is as follows.

1. Spoiler makes one of the following moves.

   (a) $\Diamond$-move: Spoiler moves pebble a from a_{i-1} to a_i. It must be the case that there is an edge (a_{i-1}, a_i) in the digraph $\mathcal{G}$.

   (b) $\Box$-move: Spoiler moves pebble b from b_{i-1} to b_i. It must be the case that there is an edge (b_{i-1}, b_i) in the digraph $\mathcal{H}$.

   (c) $\Diamond^+$-move: Spoiler moves pebble a from a_{i-1} to a_i. It must be the case that there is a non-trivial path (that is, consisting of at least one edge, though possibly a self-loop) from a_{i-1} to a_i in the digraph $\mathcal{G}$.

   (d) $\Box^+$-move: Spoiler moves pebble b from b_{i-1} to b_i. It must be the case that there is a non-trivial path from b_{i-1} to b_i in the digraph $\mathcal{H}$.

   (e) $@_n$-move: Spoiler moves the pebble a to the point $\mu(n)$ (note that this move has no dual).

2. Duplicator replies with the corresponding move.

   (a) $\Diamond$-move: Duplicator moves pebble b from b_{i-1} to b_i. It must be the case that there is an edge (b_{i-1}, b_i) in the digraph $\mathcal{H}$.

   (b) $\Box$-move: Duplicator moves pebble a from a_{i-1} to a_i. It must be the case that there is an edge (a_{i-1}, a_i) in the digraph $\mathcal{G}$.

   (c) $\Diamond^+$-move: Duplicator moves pebble b from b_{i-1} to b_i. It must be the case that there is a non-trivial path from b_{i-1} to b_i in the digraph $\mathcal{H}$.

   (d) $\Box^+$-move: Duplicator moves pebble a from a_{i-1} to a_i. It must be the case that there is a non-trivial path from a_{i-1} to a_i in the digraph $\mathcal{G}$.

(e) $@_n$-move: Duplicator moves the pebble b to the point $\lambda(n)$.

In order to determine the winner of a play of a game, the notion of equivalence on points is needed.

Definition 4.4.2. Let $(\mathcal{G}, \mu)$ and $(\mathcal{H}, \lambda)$ be two $\mathsf{P} \cup \mathsf{N}$-structures, and let u and v be points of $\mathcal{G}$ and $\mathcal{H}$, respectively. It is said that u and v are *equivalent*, written $u \simeq v$, if for every $p \in \mathsf{P}$ and $n \in \mathsf{N}$: $u \in \mu(p)$ if, and only if, $v \in \lambda(p)$; and $u = \mu(n)$ if, and only if, $v = \lambda(n)$.

The above definition summarises the idea that only the assignments to proposition and nominal symbols effects the equivalence of two points.

Definition 4.4.3. Duplicator wins a play of the r-round $\mathrm{HGL}(\mathsf{P}, \mathsf{N})$-game on two pointed $\mathsf{P} \cup \mathsf{N}$-structures $(\mathcal{G}, \mu, u)$ and $(\mathcal{H}, \lambda, v)$ if either: at some point before the end of the play Spoiler cannot make a move (which is only the case when both pebbles are on vertices of out-degree 0 and there are no nominals); or for all $i \in \{0, 1, \ldots, r\}$, $a_i \simeq b_i$. In all other cases, Spoiler wins; in particular, Spoiler wins a play if Duplicator is unable to reply to a move of Spoiler. Duplicator has a winning strategy in the r-round $\mathrm{HGL}(\mathsf{P}, \mathsf{N})$-game on two pointed Kripke $\mathsf{P} \cup \mathsf{N}$-structures if she can always reply to Spoiler's moves in any play so as to force a win for Duplicator, with Spoiler having a winning strategy if he can move so as to force at least one play of the game for which Duplicator does not win.

There is a relationship between r-round $\mathrm{HGL}(\mathsf{P}, \mathsf{N})$-games and the logical equivalence of pointed structures in the logic $\mathrm{HGL}_r(\mathsf{P}, \mathsf{N})$.

Theorem 4.4.4. *Let $r \geq 0$, let P be a set of propositional symbols and let N be a set of nominals. Let $(\mathcal{G}, \mu)$ and $(\mathcal{H}, \lambda)$ be $\mathsf{P} \cup \mathsf{N}$-structures and let u and v be points of $\mathcal{G}$ and $\mathcal{H}$, respectively. The following are equivalent.*

1. $(\mathcal{G}, \mu, u) \equiv_{\mathrm{HGL}_r(\mathsf{P},\mathsf{N})} (\mathcal{H}, \lambda, v)$.
2. *Duplicator has a winning strategy in the r-round $\mathrm{HGL}(\mathsf{P}, \mathsf{N})$-game on $(\mathcal{G}, \mu, u)$ and $(\mathcal{H}, \lambda, v)$.*

Proof. For brevity, throughout this proof: on occasion $\mathcal{C} = (\mathcal{G}, \mu)$ and $\mathcal{D} = (\mathcal{H}, \lambda)$ is written; the edge set of the digraph $\mathcal{G}$ (resp. $\mathcal{H}$) is denoted by $E^{\mathcal{G}}$ (resp. $E^{\mathcal{H}}$); and the non-reflexive transitive closure of $E^{\mathcal{G}}$ (resp. $E^{\mathcal{H}}$) is denoted by $E^{\mathcal{G}}_+$ (resp. $E^{\mathcal{H}}_+$).

Begin by noting that the result trivially holds for $r = 0$. In particular, for any point u of some Kripke $\mathsf{P} \cup \mathsf{N}$-structure $(\mathcal{G}, \mu)$, there exists a formula $\varphi^0_{\mathcal{C},u}$ of $\mathrm{HGL}_0(\mathsf{P}, \mathsf{N})$ such that for any point v of any $\mathsf{P} \cup \mathsf{N}$-structure $(\mathcal{H}, \lambda)$, $(\mathcal{G}, \mu, u) \equiv_{\mathrm{HGL}_0(\mathsf{P},\mathsf{N})} (\mathcal{H}, \lambda, v)$ if, and only if, $u \simeq v$ if, and only if, Duplicator has a winning strategy in the 0-round $\mathrm{HGL}(\mathsf{P}, \mathsf{N})$-game on $(\mathcal{G}, \mu, u)$ and $(\mathcal{H}, \lambda, v)$ if, and only if, $(\mathcal{H}, \lambda, v) \models \varphi^0_{\mathcal{C},u}$. The formula $\varphi^0_{\mathcal{C},u}$ is actually the formula

$$\bigwedge\{p : p \in \mathsf{P}, u \in \mu(p)\} \wedge \bigwedge\{\neg p : p \in \mathsf{P}, u \notin \mu(p)\} \\ \wedge \bigwedge\{n : n \in \mathsf{N}, u = \mu(n)\} \wedge \bigwedge\{\neg n : n \in \mathsf{N}, u \neq \mu(n)\}.$$

Lemma 4.4.5. *Let $r \geq 0$ and let $(\mathcal{G}, \mu)$ be a $\mathsf{P} \cup \mathsf{N}$-structure. For any point u of $\mathcal{G}$, there exists a formula $\varphi^r_{\mathcal{C},u}$ of $\mathrm{HGL}_r(\mathsf{P}, \mathsf{N})$ such that for any point v of any $\mathsf{P} \cup \mathsf{N}$-structure*

$(\mathcal{H}, \lambda)$, *Duplicator has a winning strategy in the* r*-round* $\mathrm{HGL}(\mathsf{P}, \mathsf{N})$*-game on* $(\mathcal{G}, \mu, u)$ *and* $(\mathcal{H}, \lambda, v)$ *if, and only if,* $(\mathcal{H}, \lambda, v) \models \varphi^r_{\mathcal{C},u}$.

Proof. Let the following be the induction hypothesis $\mathrm{Ind}(r)$, for some $r \geq 0$: for any point u of $\mathcal{G}$, there exists a formula $\varphi^r_{\mathcal{C},u}$ of $\mathrm{HGL}_r(\mathsf{P}, \mathsf{N})$ such that for any point v of any Kripke $\mathsf{P} \cup \mathsf{N}$-structure $(\mathcal{H}, \lambda)$, Duplicator has a winning strategy in the r-round $\mathrm{HGL}(\mathsf{P}, \mathsf{N})$-game on $(\mathcal{C}, u)$ and $(\mathcal{D}, v)$ if, and only if, $(\mathcal{D}, v) \models \varphi^r_{\mathcal{C},u}$. $\mathrm{Ind}(0)$ holds by the remark preceding the statement of the lemma.

For any point u of $\mathcal{G}$, define the formula $\varphi^{r+1}_{\mathcal{C},u}$ of $\mathrm{HGL}_{r+1}(\mathsf{P}, \mathsf{N})$ as follows:

$$\bigwedge_{(u,u') \in E^{\mathcal{G}}} \Diamond \varphi^r_{\mathcal{C},u'} \wedge \bigwedge_{(u,u') \in E^{\mathcal{G}}_+} \Diamond^+ \varphi^r_{\mathcal{C},u'} \wedge \Box \bigvee_{(u,u') \in E^{\mathcal{G}}} \varphi^r_{\mathcal{C},u'} \wedge \Box^+ \bigvee_{(u,u') \in E^{\mathcal{G}}_+} \varphi^r_{\mathcal{C},u'}$$
$$\wedge \bigwedge_{n \in \mathsf{N}} @_n \varphi^r_{\mathcal{C},\mu(n)}.$$

Suppose that: v is some point of some Kripke $\mathsf{P} \cup \mathsf{N}$-structure $(\mathcal{H}, \lambda)$; $(\mathcal{D}, v) \models \varphi^{r+1}_{\mathcal{C},u}$; and an $(r+1)$-round $\mathrm{HGL}(\mathsf{P}, \mathsf{N})$-game is played on $(\mathcal{C}, u)$ and $(\mathcal{D}, v)$.

1. Suppose that Spoiler's first move in some play is a $\Diamond$-move so that $a_1 = u'$. As $(\mathcal{D}, v) \models \varphi^{r+1}_{\mathcal{C},u}$, it follows that $(\mathcal{D}, v) \models \Diamond \varphi^r_{\mathcal{C},u'}$. So, there exists some point $v' \in \{v' \text{ is a point of } \mathcal{H} : (v, v') \in E^{\mathcal{H}}\}$ such that $(\mathcal{D}, v') \models \varphi^r_{\mathcal{C},u'}$. Duplicator replies with $b_1 = v'$. By $\mathrm{Ind}(r)$, Duplicator has a winning strategy in the r-round $\mathrm{HGL}(\mathsf{P}, \mathsf{N})$-game on $(\mathcal{C}, u')$ and $(\mathcal{D}, v')$. If Spoiler's move is a $\Diamond^+$-move then a similar argument is made.

2. Suppose that Spoiler's first move in some play is a $\Box$-move so that $b_1 = v'$. As $(\mathcal{D}, v) \models \varphi^{r+1}_{\mathcal{C},u}$, it follows that $(\mathcal{D}, v) \models \Box \bigvee_{(u,u') \in E^{\mathcal{G}}} \varphi^r_{\mathcal{C},u'}$; that is,$(\mathcal{D}, v') \models \bigvee_{(u,u') \in E^{\mathcal{G}}} \varphi^r_{\mathcal{C},u'}$. Hence, $(\mathcal{D}, v') \models \varphi^r_{\mathcal{C},u'}$, for some point $u' \in \{u' \text{ is a point of } \mathcal{G} : (u, u') \in E^{\mathcal{G}}\}$. Duplicator replies with $a_1 = u'$. By $\mathrm{Ind}(r)$, Duplicator has a winning strategy in the r-round $\mathrm{HGL}(\mathsf{P}, \mathsf{N})$-game on $(\mathcal{C}, u')$ and $(\mathcal{D}, v')$. If Spoiler's move is a $\Box^+$-move then a similar argument is made.

3. Suppose that Spoiler's first move in some play is a $@_n$-move so that $a_1 = \mu(n)$. As $(\mathcal{D}, v) \models \varphi^{r+1}_{\mathcal{C},u}$, it follows that $(\mathcal{D}, v) \models @_n \varphi^r_{\mathcal{C},\mu(n)}$; that is, $(\mathcal{D}, \lambda(n)) \models \varphi^r_{\mathcal{C},\mu(n)}$. Duplicator replies with $b_1 = \lambda(n)$. By $\mathrm{Ind}(r)$, Duplicator has a winning strategy in the r-round $\mathrm{HGL}(\mathsf{P},\mathsf{N})$-game on $(\mathcal{C}, \mu(n))$ and $(\mathcal{D}, \lambda(n))$.

Consequently, by replying with a_1 or b_1, appropriately, as in each of the cases above, Duplicator has a winning strategy in the $(r+1)$-round $\mathrm{HGL}(\mathsf{P}, \mathsf{N})$-game on $(\mathcal{C}, u)$ and $(\mathcal{D}, v)$.

Conversely, suppose that v is a point of some $\mathsf{P} \cup \mathsf{N}$-structure $(\mathcal{H}, \lambda)$ and that Duplicator has a winning strategy in the $(r+1)$-round $\mathrm{HGL}(\mathsf{P}, \mathsf{N})$-game on $(\mathcal{C}, u)$ and $(\mathcal{D}, v)$. Consider a play of the game.

1. Suppose that Spoiler's first move in some play is a $\Diamond$-move so that $a_1 = u'$. Let Duplicator's response (according to the winning strategy) be $b_1 = v'$. In particular, Duplicator has a winning strategy in the r-round $\mathrm{HGL}(\mathsf{P}, \mathsf{N})$-game on $(\mathcal{C}, u')$ and $(\mathcal{D}, v')$. By $\mathrm{Ind}(r)$, $(\mathcal{D}, v') \models \varphi^r_{\mathcal{C},u'}$ and so $(\mathcal{D}, v) \models \Diamond \varphi^r_{\mathcal{C},u'}$. Hence, as Spoiler's first

$\Diamond$-move can be arbitrary amongst all points of $\{u' \text{ is a point of } \mathcal{G} : (u,u') \in E^{\mathcal{G}}\}$, it must be that $(\mathcal{D},v) \models \bigwedge_{(u,u')\in E^{\mathcal{G}}} \Diamond\varphi^r_{\mathcal{C},u'}$ holds. Similar reasoning yields that $(\mathcal{D},v) \models \bigwedge_{(u,u')\in E^{\mathcal{G}}_+} \Diamond^+\varphi^r_{\mathcal{C},u'}$.

2. Suppose that Spoiler's first move in some play is a $\Box$-move. No matter which point of $\{v' \text{ is a point of } \mathcal{H} : (v,v') \in E^{\mathcal{H}}\}$ Spoiler moves to, so that $b_1 = v'$, there exists some point $g(v')$ of $\{u' \text{ is a point of } \mathcal{G} : (u,u') \in E^{\mathcal{G}}\}$ that Duplicator can respond with, by setting $a_1 = g(v')$, so as to ensure that Duplicator wins the subsequent r-round HGL(P,N)-game on $(\mathcal{C},g(v'))$ and $(\mathcal{D},v')$. By Ind(r), $(\mathcal{D},v') \models \varphi^r_{\mathcal{C},g(v')}$, for every point v' in $\{v' \text{ is a point of } \mathcal{H} : (v,v') \in E^{\mathcal{H}}\}$. Consequently, it must be that $(\mathcal{D},v) \models \Box \bigvee_{(u,u')\in E^{\mathcal{G}}} \varphi^r_{\mathcal{C},u'}$. Similar reasoning yields that $(\mathcal{D},v) \models \Box^+ \bigvee_{(u,u')\in E^{\mathcal{G}}_+} \varphi^r_{\mathcal{C},u'}$.

3. Suppose that Spoiler's first move in some play is a $@_n$-move, so that $a_1 = \mu(n)$. Duplicator's reply must be so that $b_1 = \lambda(n)$, and Duplicator wins the subsequent r-round HGL(P,N)-game on $(\mathcal{C},\mu(n))$ and $(\mathcal{D},\lambda(n))$. By Ind(r), $(\mathcal{D},\lambda(n)) \models \varphi^r_{\mathcal{C},\mu(n)}$ and so $(\mathcal{D},v) \models @_n\varphi^r_{\mathcal{C},\mu(n)}$; consequently, as n is an arbitrary nominal, it must be that $(\mathcal{D},v) \models \bigwedge_{n\in\mathsf{N}} @_n\varphi^r_{\mathcal{C},\mu(n)}$ holds.

Hence, $(\mathcal{D},v) \models \varphi^{r+1}_{\mathcal{C},u}$, and the lemma follows by induction. $\Box$

Return now to the proof of the main theorem. For the induction hypothesis Ind(r), where $r \geq 0$, assume that for any $\mathsf{P}\cup\mathsf{N}$-structures $(\mathcal{G},\mu)$ and $(\mathcal{H},\lambda)$ and for any points u and v of $\mathcal{G}$ and $\mathcal{H}$, respectively, the following are equivalent:

1. $(\mathcal{G},\mu,u) \equiv_{\mathrm{HGL}_r(\mathsf{P},\mathsf{N})} (\mathcal{H},\lambda,v)$.

2. Duplicator has a winning strategy in the r-round HGL(P,N)-game on $(\mathcal{G},\mu,u)$ and $(\mathcal{H},\lambda,v)$.

As remarked earlier, Ind(0) holds.

Let $(\mathcal{G},\mu)$ and $(\mathcal{H},\lambda)$ be Kripke $\mathsf{P}\cup\mathsf{N}$-structures and u and v points of $\mathcal{G}$ and $\mathcal{H}$, respectively. Suppose that $(\mathcal{C},u) \equiv_{\mathrm{HGL}_{r+1}(\mathsf{P},\mathsf{N})} (\mathcal{D},v)$; that is, for every formula $\varphi \in \mathrm{HGL}_{r+1}(\mathsf{P},\mathsf{N})$, $(\mathcal{C},u) \models \varphi$ if, and only if, $(\mathcal{D},v) \models \varphi$. Assume that Spoiler has a winning strategy for the $(r+1)$-round HGL(P,N)-game played on the Kripke $\mathsf{P}\cup\mathsf{N}$-structures $(\mathcal{C},u)$ and $(\mathcal{D},v)$. By Lemma 4.4.5, $(\mathcal{C},u) \models \varphi^{r+1}_{\mathcal{C},u}$ and $(\mathcal{D},v) \not\models \varphi^{r+1}_{\mathcal{C},u}$, with $\varphi^{r+1}_{\mathcal{C},u} \in \mathrm{HGL}_{r+1}(\mathsf{P},\mathsf{N})$. This yields a contradiction.

Conversely, suppose that $(\mathcal{C},u) \not\equiv_{\mathrm{HGL}_{r+1}(\mathsf{P},\mathsf{N})} (\mathcal{D},v)$. So, there exists a formula $\varphi \in \mathrm{HGL}_{r+1}(\mathsf{P},\mathsf{N})$ such that $(\mathcal{C},u) \models \varphi$ and $(\mathcal{D},v) \not\models \varphi$. Without loss of generality, assume that φ is of the form

$$\Diamond\psi \ ; \ \Diamond^+\psi \ ; \ \Box\psi \ ; \ \Box^+\psi \ ; \text{ or } @_n\psi,$$

where $\psi \in \mathrm{HGL}_r(\mathsf{P},\mathsf{N})$ and n is a nominal of N. To see this, note that if φ is of the form $\varphi_1 \Rightarrow \varphi_2$, for example, then $(\mathcal{D},u) \models \varphi_1$ but $(\mathcal{D},u) \not\models \varphi_2$. If $(\mathcal{C},u) \models \varphi_1$ then take φ as φ_2, otherwise take φ as φ_1.

1. Suppose that φ is of the form $\Diamond\psi$. There exists a point u' of $\mathcal{G}$ such that $(u,u') \in E^{\mathcal{G}}$ and $(\mathcal{C},u') \models \psi$. Let Spoiler's first move of an $(r+1)$-round HGL(P,N)-game on $(\mathcal{C},u)$ and $(\mathcal{D},v)$ be a $\Diamond$-move so that $a_1 = u'$. No matter which point v' of $\{v' \text{ is a point of } \mathcal{H} : (v,v') \in E^{\mathcal{H}}\}$ Duplicator replies with, $(\mathcal{D},v') \not\models \psi$. By Ind($r$),

Spoiler has a winning strategy in the subsequent r-round HGL(P, N)-game on $(\mathcal{C}, u')$ and $(\mathcal{D}, v')$. A similar argument is made when φ is of the form $\Diamond^+\psi$.

2. Suppose that φ is of the form $\Box\psi$. For every point u' of $\mathcal{G}$ such that $(u, u') \in E^{\mathcal{G}}$, it is the case that $(\mathcal{C}, u') \models \psi$. However, there is a point v' of $\{v' \text{ is a point of } \mathcal{H} : (v, v') \in E^{\mathcal{H}}\}$ such that $(\mathcal{D}, v') \not\models \psi$. Let Spoiler's first move of an $(r+1)$-round HGL(P, N)-game be a $\Box$-move so that $b_1 = v'$. No matter which point u' of $\{u' \text{ is a point of } \mathcal{G} : (u, u') \in E^{\mathcal{G}}\}$ Duplicator replies with, $(\mathcal{C}, u') \models \psi$. By Ind($r$), Spoiler has a winning strategy in the subsequent r-round HGL(P,N)-game on $(\mathcal{C}, u')$ and $(\mathcal{D}, v')$. A similar argument is made when φ is of the form $\Box^+\psi$.

3. Suppose that φ is of the form $@_n\psi$, where n is a nominal of N. Thus, $(\mathcal{C}, \mu(n)) \models \psi$ and $(\mathcal{D}, \lambda(n)) \not\models \psi$. Let Spoiler's first move of an $(r+1)$-round HGL(P, N)-game be an $@_n$-move. By Ind(r), Spoiler has a winning strategy in the subsequent r-round HGL(P, N)-game on $(\mathcal{C}, \mu(n))$ and $(\mathcal{D}, \lambda(n))$.

Thus, Spoiler has a winning strategy in the $(r+1)$-round HGL(P, N)-game played on the Kripke $\mathsf{P} \cup \mathsf{N}$-structures $(\mathcal{G}, \mu, u)$ and $(\mathcal{H}, \lambda, v)$. The result follows by induction. □

4.4.2 Games on Modal Frames

The game from the previous section is now extended so that it can be played on modal frames (graphs). Since a formula holds on a modal frame in HGL if, and only if, it is valid on that frame (see Definition 2.3.15) there are in effect unary relations representing the proposition symbols that need to be chosen, so a good starting point for building a game on frames is the Ehrenfeucht-Fraïssé game for second-order logic [Fag75] (see Section 2.1.5.3 and Definition 2.1.87).

Definition 4.4.6. Let P be a set of propositional symbols, let N be a set of nominals and let $\mathcal{G} = \langle V^{\mathcal{G}}, E^{\mathcal{G}} \rangle$ and $\mathcal{H} = \langle V^{\mathcal{H}}, E^{\mathcal{H}} \rangle$ be two modal frames. The *r-round colouring* HGL(P, N)*-game* on $\mathcal{G}$ and $\mathcal{H}$ is played by two players, the Spoiler and the Duplicator, as follows:

1. The Spoiler picks a frame (either $\mathcal{G}$ or $\mathcal{H}$). If he has chosen, say $\mathcal{G}$, he then picks $|\mathsf{P}|$ unary relations over $V^{\mathcal{G}}$ and $|\mathsf{N}|$ vertices from $V^{\mathcal{G}}$. Using these choices he then construct the valuation function $\mu : \mathsf{P} \cup \mathsf{N} \to \mathcal{P}(V^{\mathcal{G}})$, such that for each $p_i \in \mathsf{P}$, $\mu(p_i) \subseteq V^{\mathcal{G}}$ and for each $n_i \in \mathsf{N}$, $\mu(n_i) = v$, where $v \in V^{\mathcal{G}}$. He then picks a starting vertex $u \in V^{\mathcal{G}}$. On the other hand, if he had chosen frame $\mathcal{H}$, he would pick the unary relations over $V^{\mathcal{H}}$ and vertices from $V^{\mathcal{H}}$ and use these to construct the valuation function $\lambda : \mathsf{P} \cup \mathsf{N} \to \mathcal{P}(V^{\mathcal{H}})$; he would then pick a starting vertex $v \in V^{\mathcal{H}}$.

2. The Duplicator then picks $|\mathsf{P}|$ unary relations and $|\mathsf{N}|$ vertices and uses these to construct a valuation function in exactly the same way the Spoiler has already done, but in the other structure. If the Spoiler has chosen $\mathcal{G}$ and constructed the pointed structure $(\mathcal{G}, \mu, u)$ then the Duplicator constructs $(\mathcal{H}, \lambda, v)$; alternatively if the Spoiler has chosen $\mathcal{H}$ and constructed the pointed structure $(\mathcal{H}, \lambda, v)$ then the Duplicator constructs $(\mathcal{G}, \mu, u)$.

3. They then play the HGL(P, N)-game on the resulting pointed ($\mathsf{P} \cup \mathsf{N}$)-structures $(\mathcal{G}, \mu, u)$ and $(\mathcal{H}, \lambda, v)$.

The winning conditions for a play of the r-round colouring HGL(P, N)-game are analogous to those of the r-round HGL(P, N)-game. Note that the pair of moves for each player in which they select a valuation function and a start vertex are referred to as the *colouring-start moves.*

There is a relationship between r-round colouring HGL(P, N)-games and frame definability (see Definitions 2.3.15 and 2.3.16) in the logic $\mathrm{HGL}_r(\mathsf{P}, \mathsf{N})$.

Theorem 4.4.7. *Let $r \geq 0$, let P be a set of proposition symbols, let N be a set of nominals and let $\mathcal{G}$ and $\mathcal{H}$ be two frames. The following are equivalent.*

1. $\mathcal{G} \equiv_{HGL_r(\mathsf{P},\mathsf{N})} \mathcal{H}$
2. *Duplicator has a winning strategy in the r-round colouring* HGL(P, N)*-game on $\mathcal{G}$ and $\mathcal{H}$.*

Proof. Assume that $\mathcal{G} \equiv_{\mathrm{HGL}_r(\mathsf{P},\mathsf{N})} \mathcal{H}$. Suppose that Spoiler has a winning strategy in the r-round colouring HGL(P, N)-game on $\mathcal{G}$ and $\mathcal{H}$. Let Spoiler make a colouring-start move as dictated by his winning strategy; that is, Spoiler builds the $\mathsf{P} \cup \mathsf{N}$-structure $(\mathcal{H}, \lambda)$ and chooses a point v of $V^{\mathcal{H}}$. Suppose that Duplicator replies by making a colouring-start move so as to build the $\mathsf{P} \cup \mathsf{N}$-structure $(\mathcal{G}, \mu)$ and chooses the point u of $V^{\mathcal{G}}$. By hypothesis, no matter what Duplicator's colouring-start move, Spoiler wins the r-round HGL(P, N)-game on $(\mathcal{G}, \mu, u)$ and $(\mathcal{H}, \lambda, v)$. Thus, by Theorem 4.4.4, there is a formula $\varphi^r_{\mu,u}$ of $\mathrm{HGL}_r(\mathsf{P}, \mathsf{N})$ for which $(\mathcal{G}, \mu, u) \models \varphi^r_{\mu,u}$ and $(\mathcal{H}, \lambda, v) \not\models \varphi^r_{\mu,u}$. Define Φ as $\bigvee\{\varphi^r_{\mu,u} : \mu : \mathsf{P} \cup \mathsf{N} \to \mathcal{P}(V^{\mathcal{G}})$ is a valuation function, u is a point of $\mathcal{G}\}$. In particular, $\mathcal{G} \models \Phi$, and so, by hypothesis, it must be that $\mathcal{H} \models \Phi$. However, $(\mathcal{H}, \lambda, v) \not\models \Phi$, which yields a contradiction.

Conversely, suppose that $\mathcal{G} \not\equiv_{\mathrm{HGL}_r(\mathsf{P},\mathsf{N})} \mathcal{H}$. This means that there is a formula $\varphi \in \mathrm{HGL}_r(\mathsf{P}, \mathsf{N})$ such that either $\mathcal{G} \models \varphi$ and $\mathcal{H} \not\models \varphi$ or $\mathcal{G} \not\models \varphi$ and $\mathcal{H} \models \varphi$. Assume it is the former case and note that since $\mathcal{H} \not\models \varphi$ then there exists some valuation function λ and start vertex v, such that $(\mathcal{H}, \lambda, v) \not\models \varphi$. Let Spoiler make a colouring-start move of the r-round colouring HGL(P, N)-game on $\mathcal{G}$ and $\mathcal{H}$ by selecting $\mathcal{H}$ and building such a pointed ($\mathsf{P} \cup \mathsf{N}$)-structure $(\mathcal{H}, \lambda, u)$. The Duplicator replies by building some ($\mathsf{P} \cup \mathsf{N}$)-structure $(\mathcal{G}, \mu)$ and choosing some point u of $\mathcal{G}$. No matter how she has replied to his colouring-start move, it must be the case that $(\mathcal{G}, \mu, u) \models \varphi$. For the latter case, where $\mathcal{H} \models \varphi$ and $\mathcal{G} \not\models \varphi$ the same argument is made, except with the Spoiler choosing $\mathcal{G}$. By Theorem 4.4.4, Spoiler has a winning strategy in the r-round colouring HGL(P, N)-game on $\mathcal{G}$ and $\mathcal{H}$. The result follows. □

The above game on modal frames is derived from and has a lot of similarities to the Ehrenfeucht-Fraïssé style game for second-order logic, in that it takes as its input two structures and the Spoiler chooses which structure they start playing on. There exists a restriction of this game where the input is a class of structures, rather than two particular structures and the Spoiler has to start playing on one particular structure chosen by the Duplicator. This restriction is called the Ajtai-Fagin game (see Section 2.1.5.4 and

Definition 2.1.88) and it has been used to show that certain problems are not definable in various classes of monadic second-order logic (of which modal logics are a subset of), see [Sch96, Nur96, AF97]. As it turns out in the next section, such a restriction of the game on modal frames is not needed in order to obtain the required results.

4.5 Playing games in Hybrid Graph Logic

Theorem 4.4.7 provides the basic tool needed to prove expressibility results as regards the problems definable in various fragments of Hybrid Graph Logic.

First, an observation is made with regards to a winning strategy for Spoiler in an r-round HGL(P,N)-game on two pointed $(\mathsf{P}\cup\mathsf{N})$-structures $(\mathcal{G},\mu,u)$ and $(\mathcal{H},\lambda,v)$, where $\mathcal{G}$ and $\mathcal{H}$ are strongly connected digraphs having at least one edge. If Spoiler has a winning strategy then this strategy can be assumed to be such that: either a play does not involve a $\Diamond^+$-move, a $\Box^+$-move or a $@_n$-move; or a play consists of a $\Diamond^+$-move, a $\Box^+$-move or a $@_n$-move followed by (at most) $r-1$ $\Diamond$ or $\Box$-moves. To see this, note that if Spoiler's winning strategy were to involve the undertaking of a $\Diamond^+$-move, a $\Box^+$-move or a $@_n$-move mid-way through a play then Spoiler may as well have made this move as the first move, given that any configuration in any play (on some modal frame) is completely determined by the point on which the two pebbles sit (that is, there is no 'history' associated with the configuration) and any vertex is reachable from any other vertex via a non-trivial path in the respective digraph. This observation helps in the upcoming constructions of strategies for Duplicator in response to Spoiler, in that only Duplicator's response to plays by Spoiler of the above form need be considered.

Given this above observation, it can be exactly characterised when the Duplicator has a winning strategy in the r-round HGL(P,N)-game on two pointed (P,N)-structures $(\mathcal{G},\mu,u)$ and $(\mathcal{H},\lambda,v)$.

Lemma 4.5.1. *Let $(\mathcal{G},\mu,u)$ and $(\mathcal{H},\lambda,v)$ be two pointed (P,N)-structures that are strongly connected and have a least one edge. The following are equivalent:*

1. *The Duplicator wins the r-round* HGL(P,N)*-game on $(\mathcal{G},\mu,u)$ and $(\mathcal{H},\lambda,v)$.*
2. *The Duplicator wins the restricted r-round* HGL(P,N)*-game on $(\mathcal{G},\mu,u)$ and $(\mathcal{H},\lambda,v)$, where the first move can be of any type and the remaining $(r-1)$ moves are either $\Diamond$-moves or $\Box$-moves.*

Proof. Assume that the Spoiler wins the restricted game. He can simply apply the same strategy to win the unrestricted game.

Assume that the Duplicator wins the restricted game. The strategy she uses to win gives rise to a pair of functions $s_{\mathcal{G}} : G \mapsto H$ and $s_{\mathcal{H}} : H \mapsto G$, which are constructed from her response to Spoiler's initial move: if Spoiler makes a $\Diamond^+$-move to a_1 in $\mathcal{G}$, then Duplicator responds in $\mathcal{H}$ with $s_{\mathcal{G}}(a_1)$ and if Spoiler makes a $\Box^+$-move to b_1 in $\mathcal{H}$, then Duplicator responds in $\mathcal{G}$ with $s_{\mathcal{H}}(b_1)$. Observe that since the structures $\mathcal{G}$ and $\mathcal{H}$ are strongly connected, the Spoiler's initial $\Diamond^+$ or $\Box^+$ move can be to any vertex and so the functions $s_{\mathcal{G}}$ and $s_{\mathcal{H}}$ are total.

Now, in the unrestricted game the Duplicator can use the functions $s_{\mathcal{G}}$ and $s_{\mathcal{H}}$ to decide where to move in response to a $\Diamond^+$ or $\Box^+$ move by Spoiler; she can do this even if such

a move is made part way through a play because, as mentioned above, the functions are total. Any $\Diamond$ or $\Box$ moves that follow a $\Diamond^+$ or $\Box^+$ move are responded to by the Duplicator using the same strategy as was used to win the restricted game. Hence she can win the unrestricted game.

If the Spoiler makes an $@_n$-move at any point, the Duplicator does not have a choice in her replying move, but this does not pose a problem as she has a winning strategy in the restricted game that starts with any $@_n$-move and can use the $s_{\mathcal{G}}$ and $s_{\mathcal{H}}$ functions to win a play that involves $\Diamond^+$ or $\Box^+$ moves after an $@_n$-move. $\Box$

4.5.1 Variable quantifier-rank

The games start by focussing on hierarchies obtained by fixing the number of proposition symbols and nominals and allowing the quantifier-rank to vary.

Consider the situation where there are no propositional symbols or nominals available.

Theorem 4.5.2. *Let* $r \geq 0$. $\mathrm{HGL}_r(\emptyset, \emptyset) \subset \mathrm{HGL}_{r+1}(\emptyset, \emptyset)$.

Proof. Let $r \geq 1$ and denote the directed path of length r by $\mathcal{P}_r$. Consider the r-round colouring $\mathrm{HGL}_r(\emptyset, \emptyset)$-game on the frames $\mathcal{P}_r$ and $\mathcal{P}_{r+1}$. Note that the colouring-start round consists of just selecting a starting vertex.

Suppose that Spoiler chooses $\mathcal{P}_r$ and selects the starting vertex u. Duplicator now responds in $\mathcal{P}_{r+1}$ with the vertex v such that the distance from v to the last vertex in $\mathcal{P}_{r+1}$ equals the distance from u to the last vertex in $\mathcal{P}_r$. In the following r moves of play the Duplicator can ensure that the distance from each of the pebbles a_i and b_i to the last vertex is equal in both structures and hence wins the game. The reason she can maintain this property is that the only position where it does not hold is when the pebbles are located on the first vertex of each path; a situation that cannot arise since the paths are directed (i.e. pebbles always move *away* from the first vertex of each path) and her strategy ensures that she does not place a pebble on the first vertex of $\mathcal{P}_{r+1}$.

Alternatively, suppose that Spoiler chooses $\mathcal{P}_{r+1}$ and the starting vertex v. Duplicator's strategy is again to ensure that the distance from the pebbles to the ends of the paths is equal in both structures. She can achieve this for any starting point the Spoiler chooses of $\mathcal{P}_{r+1}$ *except* if it is the first vertex on the path. In this situation she responds with the point u which is the first vertex on the path $\mathcal{P}_r$. Now considering this situation, if Spoiler's first move is a $\Diamond^+$-move then Duplicator can respond by moving the pebble in $\mathcal{P}_{r+1}$ to a vertex such that the distance between a_1 and the end of $\mathcal{P}_r$ equals the distance between b_1 and the end of $\mathcal{P}_{r+1}$ and hence she wins.

Next, consider the situation where the Spoiler's first move is a $\Box^+$-move and say that the Spoiler moves the pebble from b_0 a distance of l along the path $\mathcal{P}_{r+1}$ to b_1. If $l > 1$ then the Duplicator responds by moving the pebble from a_0 a distance of $l - 1$ along the path $\mathcal{P}_r$ to a_1. The distance from a_1 to the end of $\mathcal{P}_r$ is equal to the distance from b_1 to the end of $\mathcal{P}_{r+1}$ and hence she wins. If $l = 1$ then the Duplicator responds with a_1 a distance of 1 along the path $\mathcal{P}_r$. The distances from a_1 and b_1 to the ends of their paths are not equal (they are $r - 1$ and r respectively) and the Spoiler can force the Duplicator to remain in this state as long as he makes $\Diamond$ or $\Box$-moves. If he made a $\Diamond^+$-move such that the distance from a_i to a_{i+1} is m, then the Duplicator moves such that the distance from

b_i to b_{i+1} is $(m+1)$. If he made a $\Box^+$-move such that the distance from b_i to b_{i+1} is m, where $m \geq 2$, then Duplicator moves such that the distance from a_i to a_{i+1} is $(m-1)$; the case where $m = 1$ is the same as for a $\Box$-move (i.e. the Spoiler maintains the difference of 1 in the distances from the pebbles to the last vertices). But, if Spoiler does play any combination of $(r-1)$ $\Diamond$ or $\Box$-moves in then a_r will be on the last vertex of $\mathcal{P}_r$ and b_r will be on the penultimate vertex of $\mathcal{P}_{r+1}$ and so the Duplicator wins.

The case where Spoiler's first move is a $\Diamond$ or $\Box$-move is covered by the previous case for a $\Box^+$-move where $l = 1$ (since the Duplicator responds with a move to an adjacent vertex) and hence the Duplicator wins the r-round colouring $\mathrm{HGL}_r(\emptyset, \emptyset)$-game on $\mathcal{P}_r$ and $\mathcal{P}_{r+1}$.

However, Spoiler clearly has a winning strategy in the $(r+1)$-round colouring $\mathrm{HGL}_r(\emptyset, \emptyset)$-game on $\mathcal{P}_r$ and $\mathcal{P}_{r+1}$: Spoiler chooses v in a colouring-start move as the first vertex of $\mathcal{P}_{r+1}$. No matter how Duplicator replies, Spoiler wins the subsequent game by making $r+1$ $\Box$-moves. This can be seen by examining the penultimate configuration of the game: a_r will be the last vertex of $\mathcal{P}_r$ and b_r will be the penultimate vertex of $\mathcal{P}_{r+1}$ and in the final round the Duplicator cannot respond to Spoiler's $\Box$-move and so she loses.

The result follows by noting that only the problem consisting of every digraph and the problem consisting of no digraphs can be defined by formulae of $\mathrm{HGL}_0(\emptyset, \emptyset)$. $\Box$

In fact, the Spoiler's winning strategy gives rise to a formula that is true on $\mathcal{P}_r$ but not on $\mathcal{P}_{r+1}$. Let $\varphi_{r+1} = \neg\Diamond\Diamond \ldots \Diamond\top$, where $\Diamond$ is repeated $r+1$-times. Clearly, $\mathcal{P}_r \models \varphi_{r+1}$ as no matter which vertex in $\mathcal{P}_r$ is chosen as the start vertex is it impossible to take an $r+1$-edge walk. However, by choosing the first point of $\mathcal{P}_{r+1}$ to start from it is possible to take an $r+1$-edge walk; thus, $\mathcal{P}_{r+1} \not\models \varphi_{r+1}$.

Attention is now turned to the situation where there are no propositional symbols available but there is at least 1 nominal. For any $m \geq 1$, denote the directed cycle of length m by $\mathcal{C}_m$.

Lemma 4.5.3. *Let $r \geq 1$ and $d \geq 1$, and define $m = d(r+1)$. It is the case that $\mathcal{C}_{m+1} \equiv_{\mathrm{HGL}_r(0,d)} \mathcal{C}_{m+2}$.*

Proof. Let $\mathsf{N} = \{n_0, n_1, \ldots, n_{d-1}\}$ be a set of $d \geq 1$ nominals. For brevity, denote $\mathcal{C}_{m+1}$ by $\mathcal{G} = \langle V^{\mathcal{G}}, E^{\mathcal{G}} \rangle$ and $\mathcal{C}_{m+2}$ by $\mathcal{H} = \langle V^{\mathcal{H}}, E^{\mathcal{H}} \rangle$. Consider the r-round colouring $\mathrm{HGL}(\emptyset, \mathsf{N})$-game on $\mathcal{G}$ and $\mathcal{H}$.

Suppose that in the colouring-start phase Spoiler chooses the valuation function $\mu : \mathsf{N} \to V^{\mathcal{G}}$ and the point u. Suppose that in the first instance $\mu(n_0) = \mu(n_1) = \ldots = \mu(n_{d-1})$. Duplicator chooses the valuation function $\lambda : \mathsf{N} \to V^{\mathcal{H}}$ so that $\lambda(n_0) = \lambda(n_1) = \ldots = \lambda(n_{d-1})$ and chooses v so that the length of the path from v to $\lambda(n_0)$ in $\mathcal{H}$ is equal to the length of the path from u to $\mu(n_0)$ in $\mathcal{G}$. Duplicator clearly has a winning strategy in the subsequent r-round $\mathrm{HGL}(\emptyset, \mathsf{N})$-game on $(\mathcal{G}, \mu, u)$ and $(\mathcal{H}, \lambda, v)$. The only remark to make is that if Spoiler's first move is a $\Box^+$-move so that b_1 is such that there is an edge from $\lambda(n_0)$ to b_1 (that is, the path from b_1 to $\lambda(n_0)$ has length $m+1$) then Duplicator replies so that there is an edge from $\mu(n_0)$ to a_1 (that is, the path from a_1 to $\mu(n_0)$ has length m). Duplicator can always successfully respond to Spoiler's subsequent $r-1$ $\Diamond$- or $\Box$-moves.

Alternatively, suppose that in the colouring-start phase Spoiler begins by choosing the valuation function $\lambda : \mathsf{N} \to V^{\mathcal{H}}$ and the point v. Similarly to as above, suppose that $\lambda(n_0) = \lambda(n_1) = \ldots = \lambda(n_{d-1})$. Let the length of the path from v to $\lambda(n_0)$ be l. Duplicator chooses the valuation function $\mu : \mathsf{N} \to V^{\mathcal{G}}$ so that $\mu(n_0) = \mu(n_1) = \ldots = \mu(n_{d-1})$, and she chooses u so that the length of the path from u to $\mu(n_0)$ in $\mathcal{G}$ is $\min\{l, m\}$. Clearly (as in the previous paragraph), Duplicator has a winning strategy in the subsequent r-round HGL$(\emptyset, \mathsf{N})$-game on $(\mathcal{G}, \mu, u)$ and $(\mathcal{H}, \lambda, v)$.

Now suppose that in the colouring-start phase Spoiler chooses the valuation function $\mu : \mathsf{N} \to V^{\mathcal{G}}$ and the point u where w.l.o.g.: $\lambda(n_0) \neq \lambda(n_1)$ (and so $d \geq 2$); the nominals appear on the cycle $\mathcal{G}$ in the order $n_0, n_1, \ldots, n_{d-1}$ (consecutive nominals might sit on the same vertex); and u lies on the path from $\lambda(n_0)$ to $\lambda(n_1)$ in $\mathcal{G}$ but where $u \neq \lambda(n_1)$ (it might be the case that $u = \lambda(n_0)$, though). Duplicator replies by choosing the valuation function $\lambda : \mathsf{N} \to V^{\mathcal{H}}$ and point v as follows. Duplicator takes a copy of $(\mathcal{G}, \mu, u)$ and 'splits' an edge by replacing it with a path of length 2, with the edge to be split chosen as we now describe:

1. if the length of the path from $\mu(n_0)$ to $\mu(n_1)$ is at most $r + 2$ then there must be some n_i, where $i \neq 0$, so that the length of the path from $\mu(n_i)$ to $\mu(n_{i+1})$ (with addition modulo d) is at least $r + 2$; choose the first edge on this path to split

2. if the length of the path from $\mu(n_0)$ to $\mu(n_1)$ is at least $r + 3$ and $v = \mu(n_0)$ then split the first edge of this path

3. if the length of the path from $\mu(n_0)$ to $\mu(n_1)$ is at least $r + 3$ and $v \neq \mu(n_0)$ then split the first edge of the path from $\mu(n_0)$ to v.

The pointed structure $(\mathcal{H}, \lambda, v)$ is obtained from this 'split-edge' structure by setting $\lambda(n_i) = \mu(n_i)$, for $i \in \{0, 1, \ldots, d-1\}$, and renaming u to v. The three different constructions can be visualized in Figure 4.5.1 (where the grey vertex is the 'new' vertex and the white vertex is u or v).

It is not difficult to see that Duplicator has a winning strategy in the subsequent r-round HGL$(\emptyset, \mathsf{N})$-game on $(\mathcal{G}, \mu, u)$ and $(\mathcal{H}, \lambda, v)$. The only remark to make is that if Spoiler's first move is a $\Box^+$-move so that b_1 is the 'new' point, in $\mathcal{H}$, introduced by splitting the edge (x, y) of $\mathcal{G}$, then a_1 is chosen to be the point y of $\mathcal{G}$.

Alternatively, suppose that Spoiler begins the colouring-start phase by choosing the valuation function $\lambda : \mathsf{N} \to V^{\mathcal{H}}$ and the point v where w.l.o.g.: $\lambda(n_0) \neq \lambda(n_1)$ (and so $d \geq 2$); the nominals appear on the cycle $\mathcal{H}$ in the order $n_0, n_1, \ldots, n_{d-1}$ (consecutive nominals might sit on the same vertex); and v lies on the path from $\lambda(n_0)$ to $\lambda(n_1)$ in $\mathcal{H}$ but where $v \neq \lambda(n_1)$ (it might be the case that $v = \lambda(n_0)$, though). Duplicator chooses the valuation function $\mu : \mathsf{N} \to V^{\mathcal{G}}$ and point u as follows. Duplicator takes a copy of $(\mathcal{H}, \lambda, v)$ and 'merges' a path of length 2 by replacing it with an edge, with the path to be merged chosen as we now describe:

1. if the length of the path from $\lambda(n_0)$ to $\lambda(n_1)$ is at most $r + 2$ then there must be some n_i, where $i \neq 0$, so that the length of the path from $\lambda(n_i)$ to $\lambda(n_{i+1})$ (with addition modulo d) is at least $r + 2$; choose the first 2 edges of this path to merge (so that the point common to both edges is removed)

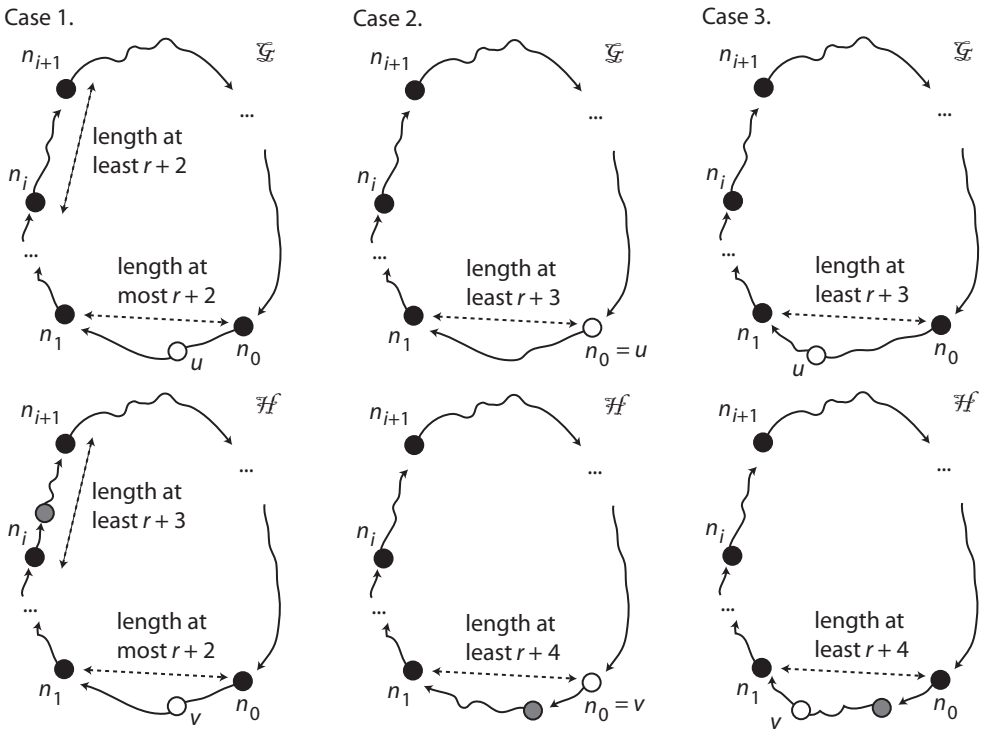

Figure 4.1: Building $(\mathcal{H}, \lambda, v)$ from $(\mathcal{G}, \mu, u)$.

2. if the length of the path from $\lambda(n_0)$ to $\lambda(n_1)$ is at least $r+3$ and either $v = \lambda(n_0)$ or there is an edge from $\lambda(n_0)$ to v then choose the last 2 edges of this path to merge

3. if the length of the path from $\lambda(n_0)$ to $\lambda(n_1)$ is at least $r+3$ and $v \neq \lambda(n_0)$ and there is no edge from $\lambda(n_0)$ to v then choose the first 2 edges of this path to merge.

The pointed structure $\langle\langle \mathcal{G}, \mu\rangle, u\rangle$ is obtained from this 'merge-edge' structure by setting $\mu(n_i) = \lambda(n_i)$, for $i \in \{0, 1, \ldots, d-1\}$, and renaming v to u. The different constructions can be visualized in Figure 4.5.1.

Again, it is not difficult to see that Duplicator has a winning strategy in the subsequent r-round HGL$(\emptyset, \mathsf{N})$-game on $(\mathcal{G}, \mu, u)$ and $(\mathcal{H}, \lambda, v)$. Consequently, the result follows from Theorem 4.4.7. □

Lemma 4.5.4. *Let $r \geq 1$ and $d \geq 1$, and define $m = d(r+1)$. There exists a formula φ of HGL$_{r+1}(0, d)$ such that $C_{m+1} \models \varphi$ and $\mathcal{C}_{m+2} \not\models \varphi$.*

Proof. Let $\mathsf{N} = \{n_0, n_1, \ldots, n_{d-1}\}$ be a set of $d \geq 1$ nominals. For brevity, denote $\mathcal{C}_{m+1}$ by $\mathcal{G} = \langle V^{\mathcal{G}}, E^{\mathcal{G}}\rangle$ and $\mathcal{C}_{m+2}$ by $\mathcal{H} = \langle V^{\mathcal{H}}, E^{\mathcal{H}}\rangle$. We shall prove that Spoiler has a winning strategy in the $(r+1)$-round colouring HGL$(\emptyset, \mathsf{N})$-game on $\mathcal{G}$ and $\mathcal{H}$. The result then follows by Theorem 4.4.7.

Consider a play in the $(r+1)$-round colouring HGL$(\emptyset, \mathsf{N})$-game on $\mathcal{G}$ and $\mathcal{H}$. Suppose that $d > 1$. In Spoiler's colouring-start move: Spoiler chooses $\lambda : \mathsf{N} \to V^{\mathcal{H}}$ such that the nominals $n_0, n_1, \ldots, n_{d-1}$ appear in order on the directed cycle $\mathcal{H}$ and so that: the length of the path from $\lambda(n_0)$ to $\lambda(n_1)$ is $r+3$; for $i = 1, 2, \ldots, d-2$, the length of the path from the point $\lambda(n_i)$ to the point $\lambda(n_{i+1})$ is $r+1$; and the length of the path from the point $\lambda(n_{d-1})$ to the point $\lambda(n_0)$ is $r+1$. In addition, Spoiler chooses v as the point x for which there is an edge $(\lambda(n_0), x)$.

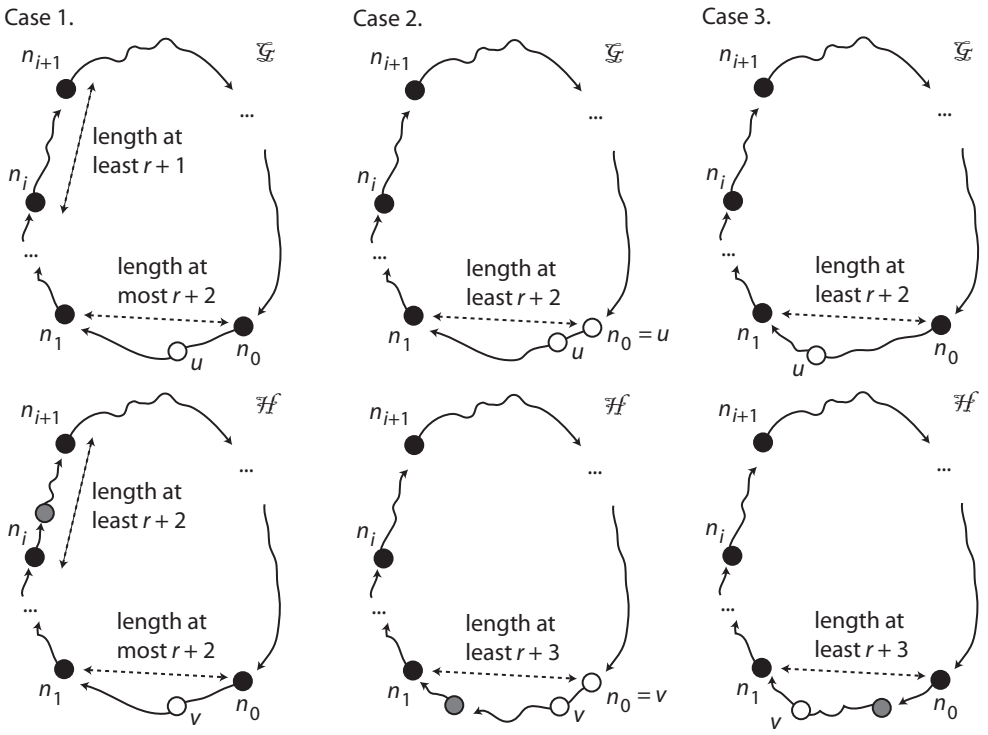

Figure 4.2: Building $(\mathcal{G}, \mu, u)$ from $(\mathcal{H}, \lambda, v)$.

Consider Duplicator's response if she wishes to progress to a win. No matter which valuation function μ and point u Duplicator chooses, the path of length $r+1$ starting at u in $(\mathcal{G}, \mu, u)$ must be nominal-free (otherwise, Spoiler would walk along the path starting at v in $\mathcal{H}$ via $r+1$ $\Box$-moves). Hence, there are two nominals, n_i and n_j, say, so that there is a nominal-free path from $\mu(n_i)$ to $\mu(n_j)$ in $\langle\langle \mathcal{G}, \mu\rangle, u\rangle$ of length at least $r+3$. Consequently, there are two nominals, $n_{i'}$ and $n_{j'}$, say, so that there is a nominal-free path from $\mu(n_{i'})$ to $\mu(n_{j'})$ in $\mathcal{G}$ of length at at most r (recall, $\mathcal{G}$ is a cycle of length $d(r+1)+1$ and $d \geq 2$). Spoiler now makes a $@_{n_{i'}}$-move so that $a_1 = \mu(n_{i'})$, with Duplicator necessarily having to ensure that $b_1 = \lambda(n_{i'})$. Spoiler now makes r $\Diamond$-moves which results in him winning the play.

Suppose that $d = 1$. Spoiler proceeds as follows. After choosing $\lambda(n_0)$, Spoiler chooses v as the point x where there is an edge $(\lambda(n_0), x)$. No matter how Duplicator replies, when Spoiler subsequently makes $r+1$ $\Diamond$-moves, Spoiler wins the play. Hence, Spoiler has a winning strategy in the $(r+1)$-round colouring HGL$(\emptyset, \mathsf{N})$-game on $\mathcal{G}$ and $\mathcal{H}$. The result follows. $\Box$

The proof of Lemma 4.5.4 is via establishing a winning strategy for Spoiler. However, this result can also be obtained by presenting an explicit formula of $\mathrm{HGL}_{r+1}(0, d)$ that tells C_{m+1} and $\mathcal{C}_{m+2}$ apart. It is argued that in this case showing a winning strategy for the Spoiler was quicker and more intuitive, especially within the domain of game playing. Nevertheless, what follows is the construction of an explicit formula of $\mathrm{HGL}_{r+1}(0, d)$ that can tell $\mathcal{C}_{m+1}$ and $\mathcal{C}_{m+2}$ apart.

- Define χ as $\neg n_0 \wedge \neg n_1 \wedge \ldots \wedge \neg n_{d-1}$. So, χ holds at some point of some structure if no nominal sits on this point.

- Define η as $\bigwedge_{i\neq j}(\neg n_i \vee \neg n_j)$, where $i, j \in \{0, 1, \ldots, d-1\}$. So, η holds at some point of some structure if at most one nominal sits on this point.
- Define φ_0 as χ and for $i \geq 1$, define φ_i as $\chi \wedge \Diamond(\varphi_{i-1})$. So, φ_i holds at some point of some structure if there is a path of length i from this point so that no nominal sits on any point of this path.
- Define ψ_1 as $\Diamond\chi$ and for $i \geq 2$, define ψ_i as $\Diamond(\chi \wedge \psi_{i-1})$. So, ψ_i holds at some point of some structure if there is a path of length i from this point so that no nominal sits on any point of this path apart from possibly the first.
- If $d = 1$, $r \geq 1$ and $m = r + 1$ then it holds that $\mathcal{C}_{m+1} \models \neg\varphi_{r+1}$ but $\mathcal{C}_{m+2} \not\models \neg\varphi_{r+1}$.
- Define Φ as $\eta \wedge \varphi_{r+1} \wedge \bigwedge_{i=0}^{d-1} @_{n_i}\psi_r$. If $d \geq 2$, $r \geq 1$ and $m = d(r + 1)$ then we have that $\mathcal{C}_{m+1} \models \neg\Phi$ but $\mathcal{C}_{m+2} \not\models \neg\Phi$.

Note that $\Diamond^+$ and $\Box^+$ have not been used in the construction of the above formula.

The following lemma is immediate.

Lemma 4.5.5. *Let $c \geq 0$ and $d \geq 0$. Let $\mathcal{G}$ be the frame with one point and no edges, and let $\mathcal{H}$ be the frame with one point and a self-loop. It is the case that $\mathcal{G} \equiv_{HGL_0(c,d)} \mathcal{H}$ but that $\mathcal{G} \models \neg\Diamond\top$ and $\mathcal{H} \not\models \neg\Diamond\top$.*

Lemmas 4.5.3, 4.5.4 and 4.5.5 yield the following result.

Theorem 4.5.6. *When $r \geq 0$ and $d \geq 1$, it is the case that $HGL_r(0, d) \subset HGL_{r+1}(0, d)$.*

Now the attention is turned to the cases when there is access to proposition symbols as well as nominals. Denote a path ρ in some digraph from vertex s to vertex t by $\rho(s, t)$. For $i \geq 1$ and $j \geq 0$ and define the digraph $\mathcal{A}_{i,j}$ as follows:

- take the disjoint union of i directed paths of length j, namely the paths $\rho_1(s_1, t_1)$, $\rho_2(s_2, t_2), \ldots, \rho_i(s_i, t_i)$ (where if $j = 0$ then $s_1 = t_1, s_2 = t_2, \ldots, s_i = t_i$)
- include also the distinct vertex s along with the edges of $\{(s, s_l), (t_l, s) : l = 1, 2, \ldots, i\}$.

The digraph $\mathcal{A}_{i,j}$ can be visualized as in Figure 4.3.

Lemma 4.5.7. *Let $r \geq 1$, $c \geq 1$ and $d \geq 0$. Define $m = d + 2^{c(r+1)}$. It is the case that $\mathcal{A}_{m-1,r} \equiv_{\mathrm{HGL}_r(c,d)} \mathcal{A}_{m,r}$.*

Proof. For brevity, denote $\mathcal{A}_{m-1,r}$ by $\mathcal{G} = \langle V^{\mathcal{G}}, E^{\mathcal{G}}\rangle$ and denote $\mathcal{A}_{m,r}$ by $\mathcal{H} = \langle V^{\mathcal{H}}, E^{\mathcal{H}}\rangle$. A path ρ_i will be denoted $\rho_i^{\mathcal{G}}$ or $\rho_i^{\mathcal{H}}$ depending upon whether it lies in $\mathcal{G}$ or $\mathcal{H}$, respectively, and the same goes for vertices (so, there is the vertex $s^{\mathcal{G}}$ of $V^{\mathcal{G}}$, the vertex $s^{\mathcal{H}}$ of $V^{\mathcal{H}}$, and so on).

Let P be a set of c propositional symbols and let N be a set of d nominals. Consider a play of the r-round colouring $\mathrm{HGL}(\mathsf{P}, \mathsf{N})$-game on $\mathcal{G}$ and $\mathcal{H}$. Suppose that Spoiler makes a colouring-start move so as to build the pointed $\mathsf{P} \cup \mathsf{N}$-structure $(\mathcal{H}, \lambda, v)$. This results in each path $\rho_i^{\mathcal{H}}$ having a specific *colour-type*: an ordered list of $r + 1$ subsets of $\mathsf{P} \cup \mathsf{N}$,

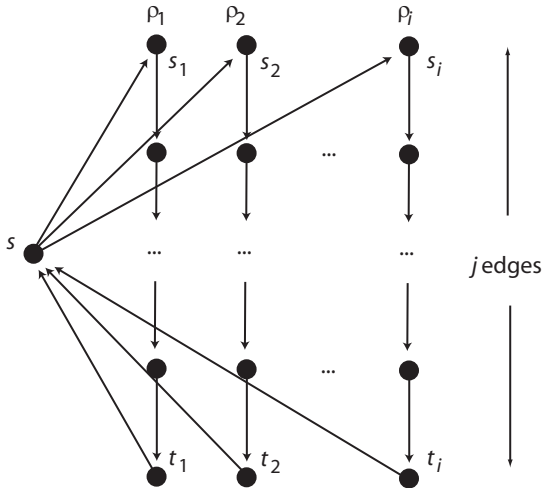

Figure 4.3: The digraph $\mathcal{A}_{i,j}$

one for each vertex w of $\rho_i^{\mathcal{H}}$, detailing for which $p \in \mathsf{P}$ or $n \in \mathsf{N}$ it holds that $w \in \lambda(p)$ or $w \in \lambda(n)$. A path $\rho_i^{\mathcal{H}}$ is called *clean* if no nominal is involved in any of the $r+1$ sets of the colour-type of $\rho_i^{\mathcal{H}}$, and *dirty* otherwise; the definition of clean and dirty is extended to colour-types also. Without loss of generality, it may be assumed that none of the paths of $\{\rho_{d+1}^{\mathcal{H}}, \rho_{d+2}^{\mathcal{H}}, \ldots, \rho_m^{\mathcal{H}}\}$ are dirty. The following assumptions can also be made.

1. If two clean paths from $\{\rho_1^{\mathcal{H}}, \rho_2^{\mathcal{H}}, \ldots, \rho_m^{\mathcal{H}}\}$ have the same colour-type then the path $\rho_m^{\mathcal{H}}$ has the same colour-type as some clean path from $\{\rho_1^{\mathcal{H}}, \rho_2^{\mathcal{H}}, \ldots, \rho_{m-1}^{\mathcal{H}}\}$.

2. If no two clean paths have the same colour-type then the path $\rho_m^{\mathcal{H}}$ is the path of colour-type $S, S, \ldots, S$, where $S = \{p \in \mathsf{P} : s^{\mathcal{H}} \in \lambda(p)\}$ (note that in this case: every path of $\{\rho_i^{\mathcal{H}} : i = 1, 2, \ldots, d\}$ must be dirty; every path of $\{\rho_i^{\mathcal{H}} : i = d+1, d+2, \ldots, m\}$ must be clean; and $s \neq \mu(n)$, for all $n \in \mathsf{N}$).

Consider $\mathcal{H}$ with the vertices of the path $\rho_m^{\mathcal{H}}$ removed (along with any incident edges); that is, $\mathcal{H} \setminus \{\rho_m^{\mathcal{H}}\}$. Let $\lambda' : \mathsf{P} \cup \mathsf{N} \to V^{\mathcal{H}} \setminus \{w : w \text{ is a vertex of } \rho_m^{\mathcal{H}}\}$ be defined via $\lambda'(p) = \lambda(p) \setminus \{w : w \text{ is a vertex of} \rho_m^{\mathcal{H}}\}$, for $p \in \mathsf{P}$, and $\lambda'(n) = \lambda(n)$, for $n \in \mathsf{N}$. For simplicity, denote the $\mathsf{P} \cup \mathsf{N}$-structure $(\mathcal{H} \setminus \{\rho_m^{\mathcal{H}}\}, \lambda')$ by $(\mathcal{H} \setminus \{\rho_m^{\mathcal{H}}\}, \lambda)$. Duplicator replies with a colouring-start move so as to build the $\mathsf{P} \cup \mathsf{N}$-structure $(\mathcal{G}, \mu)$ such that $(\mathcal{G}, \mu)$ is isomorphic to $(\mathcal{H} \setminus \{\rho_m^{\mathcal{H}}\}, \lambda)$, via the natural isomorphism $f : \mathcal{G} \to \mathcal{H} \setminus \{\rho_m^{\mathcal{H}}\}$. However, some care needs to be taken by Duplicator in the choice of u. If Spoiler has chosen v in the isomorphic copy of $\mathcal{G}$ in $\mathcal{H}$ (w.r.t. to the natural isomorphism f) then Duplicator chooses u in $\mathcal{G}$ according to the isomorphism f. There are other possibilities.

1. Suppose that the path $\rho_m^{\mathcal{H}}$ and the path $\rho_j^{\mathcal{H}}$, where $1 \leq j < m$, have the same (clean) colour-type. If Spoiler has chosen v to be the i^{th} vertex of $\rho_m^{\mathcal{H}}$ then w.l.o.g. it may be assumed that Spoiler has chosen v as the i^{th} vertex of $\rho_j^{\mathcal{H}}$ and Duplicator chooses u according to the natural isomorphism f.

2. Suppose that all clean paths from $\{\rho_i^{\mathcal{H}} : 1 \leq i \leq m\}$ have a unique colour-type, and so all paths of $\{\rho_i^{\mathcal{H}} : 1 \leq i \leq d\}$ must be dirty with all paths of $\{\rho_i^{\mathcal{H}} : d+1 \leq i \leq m\}$ clean. Recall that the colour-type of $\rho_m^{\mathcal{H}}$ is $S, S, \ldots, S$, where $S = \{p \in \mathsf{P} : s^{\mathcal{H}} \in \lambda(p)\}$, and that this colour-type does not appear as the colour-type of any path of

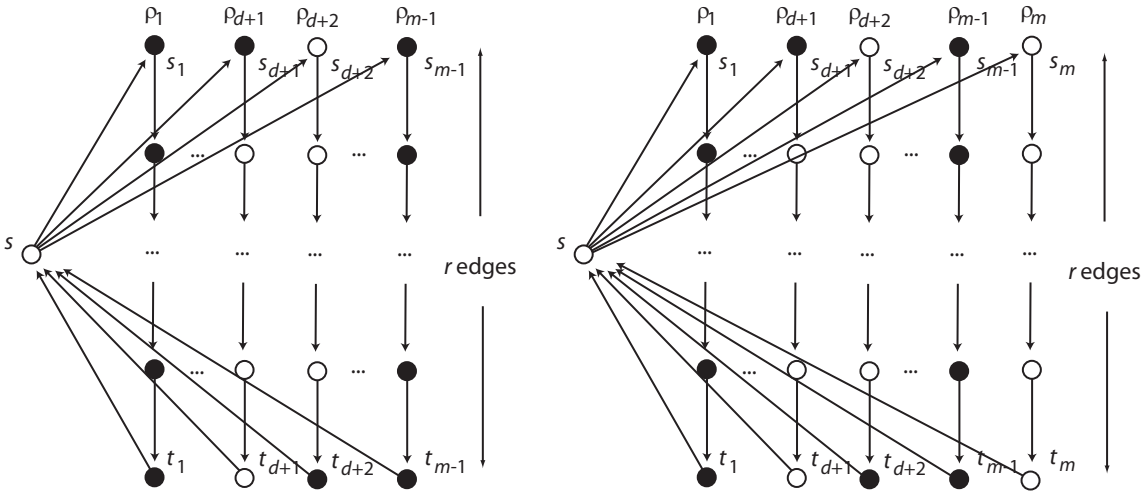

Figure 4.4: $(\mathcal{G}, \mu)$ and $(\mathcal{H}, \lambda)$ when $(\mathcal{H}, \lambda)$ has distinct clean colour-types.

$\{\rho_i^{\mathcal{G}} : 1 \leq i \leq m-1\}$. Without loss of generality, let $\rho_{d+1}^{\mathcal{H}}$ be a path whose colour-type is $S', S, \ldots, S$, for some $S' \neq S$, and let $\rho_{d+2}^{\mathcal{H}}$ be a path whose colour-type is $S, S, \ldots, S, S''$, for some $S'' \neq S$. The situation can be visualized as in Figure 4.4, where a vertex w (of $V^{\mathcal{G}}$ or $V^{\mathcal{H}}$) with the property that $\{p \in \mathsf{P} : w \in \lambda(p)\} = S$ is depicted in white.

- If Spoiler has chosen the first vertex of the path $\rho_m^{\mathcal{H}}$ as v then Duplicator chooses the second vertex of $\rho_{d+1}^{\mathcal{G}}$ as u.
- If Spoiler has chosen the i^{th} vertex of the path $\rho_m^{\mathcal{H}}$ as v, where $2 \leq i \leq r+1$, then Duplicator chooses the i^{th} vertex of $\rho_{d+1}^{\mathcal{G}}$ as u.

Following the colouring-start move, there are two essential situations.

Case 1. Suppose that the colour-type of the (clean) path $\rho_m^{\mathcal{H}}$ is the same as the colour-type of the path $\rho_j^{\mathcal{H}}$, for some path $\rho_j^{\mathcal{H}}$ where $1 \leq j < m$. Extend the natural isomorphism $f^{-1} : \mathcal{H} \setminus \{\rho_m^{\mathcal{H}}\} \to \mathcal{G}$, to the map $g : \mathcal{H} \to \mathcal{G}$ so that if w is some point of $\rho_m^{\mathcal{H}}$ then $g(w) = f^{-1}(w')$, where w' is the point of $\rho_j^{\mathcal{H}}$ analogous to w (that is, if w is the i^{th} vertex of $\rho_m^{\mathcal{H}}$ then w' is the i^{th} vertex of $\rho_j^{\mathcal{H}}$). Duplicator's strategy in the subsequent r-round HGL(P, N)-game on $(\mathcal{G}, \mu, u)$ and $(\mathcal{H}, \lambda, v)$ is simply to play according to g, if Spoiler plays in $\mathcal{H}$, or according to f, if Spoiler plays in $\mathcal{G}$. Clearly, this yields a winning strategy for Duplicator in the r-round colouring HGL(P, N)-game on $\mathcal{G}$ and $\mathcal{H}$.

Case 2. Suppose that all of the colour-types of the clean paths of $\{\rho_i^{\mathcal{H}} : 1 \leq i \leq m = d + 2^{c(r+1)}\}$ are distinct. As remarked earlier, it must be the case every path of $\{\rho_1^{\mathcal{H}}, \rho_2^{\mathcal{H}}, \ldots, \rho_d^{\mathcal{H}}\}$ is dirty with every path of $\{\rho_{d+1}^{\mathcal{H}}, \rho_{d+2}^{\mathcal{H}}, \ldots, \rho_m^{\mathcal{H}}\}$ clean. By construction: $\rho_m^{\mathcal{H}}$ has colour-type $S, S, \ldots, S$; $\rho_{d+1}^{\mathcal{H}}$ has colour-type $S', S, \ldots, S$; and $\rho_{d+1}^{\mathcal{H}}$ has colour-type $S, S, \ldots, S, S''$ (with S, S' and S'' defined as above).

Consider Spoiler's moves in a play of the r-round HGL(P, N)-game on $(\mathcal{G}, \mu, u)$ and $(\mathcal{H}, \lambda, v)$. Recall that it can be assumed that if Spoiler makes a $\Diamond^+$-move, a $\Box^+$-move or a $@_n$-move then this is will be the first move of the play and every subsequent move will be a $\Diamond$-move

or a $\Box$-move (indeed, it may be assumed that no $@_n$-move is ever made by Spoiler as Spoiler can incorporate such a move into the choice of v). Note also that Duplicator's reply to a move of Spoiler involves no choice unless either: the pebble Duplicator is moving happens to lie on $s^{\mathcal{G}}$ or $s^{\mathcal{H}}$; or Spoiler's move is a $\Diamond^+$-move or a $\Box^+$-move, either of which can only happen if the move is the first move of the play. If Spoiler's (first) move is a $\Diamond^+$-move then Duplicator simply plays according to the natural isomorphism f.

The strategy for Duplicator will be such as to force that whenever pebble a is on $s^{\mathcal{G}}$, it holds that pebble b is on $s^{\mathcal{H}}$, and *vice versa*, apart from the possibility that after the r^{th} move of the play pebble a is on $s^{\mathcal{G}}$ and pebble b not on $s^{\mathcal{H}}$. If it is the case that pebble a is on $s^{\mathcal{G}}$ and pebble b is on $s^{\mathcal{H}}$, and less than r moves have been made, then unless Spoiler makes a $\Box$-move or a $\Box^+$-move to a vertex of $\rho_m^{\mathcal{H}}$, Duplicator plays according to the natural isomorphism f. If Spoiler makes a $\Box$-move and places pebble b on $s_m^{\mathcal{H}}$ then Duplicator replies by placing pebble a on $s_{d+2}^{\mathcal{G}}$.

Suppose that Spoiler makes a $\Box^+$-move (as the first move and irrespective of the choice of v). If Spoiler places the pebble b on a point of $\mathcal{H} \setminus \{\rho_m^{\mathcal{H}}\}$ then Duplicator places pebble a according to the natural isomorphism. If Spoiler places the pebble b on a point of $\rho_m^{\mathcal{H}}$ then Duplicator's reply is as follows: if Spoiler plays on the first vertex of $\rho_m^{\mathcal{H}}$ then Duplicator replies by playing on the second vertex of $\rho_{d+1}^{\mathcal{G}}$; and if Spoiler plays on the i^{th} vertex of $\rho_m^{\mathcal{H}}$, where $2 \leq i \leq r+1$, then Duplicator replies by playing on the i^{th} vertex of $\rho_{d+1}^{\mathcal{G}}$. Note that because all of the subsequent $r-1$ moves are $\Diamond$-moves or $\Box$-moves, Duplicator can clearly win this play of the game:

- if b_1 is the first vertex of $\rho_m^{\mathcal{H}}$ and a_1 is the second vertex of $\rho_{d+2}^{\mathcal{G}}$ then in the remainder of the play the pebbles never leave the paths $\rho_m^{\mathcal{H}}$ and $\rho_{d+2}^{\mathcal{G}}$;
- if b_1 is the i^{th} vertex of $\rho_m^{\mathcal{H}}$ and a_1 is the i^{th} vertex of $\rho_{d+1}^{\mathcal{G}}$, for some $i \in \{2, 3, \ldots, r+1\}$, then by playing as directed above (when the pebbles arrive at $s^{\mathcal{G}}$ and $s^{\mathcal{H}}$), Duplicator can win the play.

Finally, it should be noted that when u and v are as defined above, Duplicator can win the play by playing as directed above. The only slightly awkward situation is when u is the second vertex of $\rho_{d+1}^{\mathcal{G}}$, v is the first vertex of $\rho_m^{\mathcal{H}}$ and all r moves are $\Diamond$ or $\Box$-moves. However, the fact that the path in $\mathcal{G}$ consisting of the last r vertices of $\rho_{d+1}^{\mathcal{G}}$ augmented with the vertex $s^{\mathcal{G}}$ has the same colour-type as the path $\rho_m^{\mathcal{H}}$ enables Duplicator to win the play.

Consequently, Duplicator has a winning strategy in the r-round colouring $\mathrm{HGL}(\mathsf{P}, \mathsf{N})$-game on the graphs $\mathcal{A}_{m-1,r}$ and $\mathcal{A}_{m,r}$ if the Spoiler selects $\mathcal{A}_{m,r}$ to perform the colouring-start move on.

Alternatively, suppose that Spoiler's colouring-start move is so as to build the pointed $\mathsf{P} \cup \mathsf{N}$-structure $(\mathcal{G}, \mu, u)$. W.l.o.g. it may be assumed that the path $\rho_{m-1}^{\mathcal{G}}$ is clean. Duplicator builds the $\mathsf{P} \cup \mathsf{N}$-structure $(\mathcal{H}, \lambda)$ by taking a copy of $(\mathcal{G}, \mu)$ and extending it with the path $\rho_m^{\mathcal{H}}$ and edges from (resp. to) $s^{\mathcal{H}}$ to the first (resp. from the last) vertex of $\rho_m^{\mathcal{H}}$ so that the colour-type of $\rho_m^{\mathcal{H}}$ is identical to the colour-type of $\rho_{m-1}^{\mathcal{G}}$. There is a natural embedding of $(\mathcal{G}, \mu)$ in $(\mathcal{H}, \lambda)$ and the point v is taken to be the image of u under this embedding; this is the pointed $\mathsf{P} \cup \mathsf{N}$-structure $(\mathcal{H}, \lambda, v)$. Given the more complicated arguments above, it should be clear that Duplicator has a winning strategy in the

r-round colouring HGL(P,N)-game on $\mathcal{A}_{m-1,r}$ and $\mathcal{A}_{m,r}$ when the Spoiler selects $\mathcal{A}_{m-1,r}$ to perform the colouring-start move on. The result follows by Theorem 4.4.7. □

Lemma 4.5.8. *Let $r \geq 1$, $c \geq 1$ and $d \geq 0$. Define $m = d + 2^{c(r+1)}$. There exists a formula φ of* HGL$_{r+1}(c,d)$ *such that $\mathcal{A}_{m-1,r} \models \varphi$ and $\mathcal{A}_{m,r} \not\models \varphi$.*

Proof. Let P be a set of c propositional symbols and let $\mathsf{N} = \{n_0, n_1, \ldots, n_{d-1}\}$ be a set of d nominals. Assume that the Duplicator starts a $(r+1)$-move colouring HGL(P,N)-game by selecting $\mathcal{A}_{m-1,r}$ and $\mathcal{A}_{m,r}$. For brevity, denote $\mathcal{A}_{m-1,r}$ by $\mathcal{G} = \langle V^{\mathcal{G}}, E^{\mathcal{G}}\rangle$ and $\mathcal{A}_{m,r}$ by $\mathcal{H} = \langle V^{\mathcal{H}}, E^{\mathcal{H}}\rangle$.

Consider the following strategy by Spoiler in the $(r+1)$-move colouring HGL(P,N)-game on $\mathcal{G}$ and $\mathcal{H}$. Spoiler begins by choosing the valuation function $\lambda : \mathsf{P} \cup \mathsf{N} \to V^{\mathcal{H}}$ so that if $d \geq 1$ then the paths of $\{\rho_1^{\mathcal{H}}, \rho_2^{\mathcal{H}}, \ldots, \rho_d^{\mathcal{H}}\}$ are all dirty, and so that the colour-types of the paths of $\{\rho_{d+1}^{\mathcal{H}}, \rho_{d+2}^{\mathcal{H}}, \ldots, \rho_m^{\mathcal{H}}\}$ are all distinct. Moreover, if $d \geq 1$ then Spoiler chooses λ so that $\lambda(n_i)$ is the first vertex of $\rho_{i+1}^{\mathcal{H}}$, for $i = 0, 1, \ldots, d-1$. Spoiler chooses v as $s^{\mathcal{H}}$. Duplicator replies with some valuation function $\mu : \mathsf{P} \cup \mathsf{N} \to V^{\mathcal{G}}$ and in order to stand a chance of winning the play, Duplicator must clearly choose u as $s^{\mathcal{G}}$. If $d \geq 1$ then in order for Duplicator to stand a chance of winning the play, clearly it must hold that w.l.o.g. $\mu(n_i)$ is the first vertex of $\rho_{i+1}^{\mathcal{G}}$, for $i = 0, 1, \ldots, d-1$.

Consequently, the paths of $\{\rho_1^{\mathcal{G}}, \rho_2^{\mathcal{G}}, \ldots, \rho_d^{\mathcal{G}}\}$ are all dirty and the paths of $\{\rho_{d+1}^{\mathcal{G}}, \rho_{d+2}^{\mathcal{G}}, \ldots, \rho_{m-1}^{\mathcal{G}}\}$ are all clean. There must be some path $\rho_j^{\mathcal{H}}$ in $\{\rho_{d+1}^{\mathcal{H}}, \rho_{d+2}^{\mathcal{H}}, \ldots, \rho_m^{\mathcal{H}}\}$ whose colour-type does not appear amongst the colour-types of the paths of $\{\rho_{d+1}^{\mathcal{G}}, \rho_{d+2}^{\mathcal{G}}, \ldots, \rho_{m-1}^{\mathcal{G}}\}$. Spoiler next makes $r+1$ $\Box$-moves and walks along the path $\rho_j^{\mathcal{H}}$. Clearly, this results in a winning play for Spoiler. Hence, Spoiler has a winning strategy for the $(r+1)$-move colouring HGL(P,N)-game on $\mathcal{G}$ and $\mathcal{H}$. The result follows by Theorem 4.4.7. □

Just as was the case with Lemma 4.5.4, it is possible to prove Lemma 4.5.8 by constructing an explicit formula of HGL$_{r+1}(c,d)$ to tell $\mathcal{A}_{m-1,r}$ and $\mathcal{A}_{m,r}$ apart. This time though, it is much more complicated and proceeding by using our game makes life much easier. However, what follows is a description of such a formula. Essentially, given some pointed $\mathsf{P}\cup\mathsf{N}$-structure $(\mathcal{G}, \mu, u)$, the formula will be the negation of the formula Φ that says: 'from u, upon which no nominal sits, one can always move along an edge so that thereafter there is a path of length r upon which no nominal sits and whose colour-type is any colour-type involving just the propositional symbols from the set P; moreover, from u, one can move to a vertex on which exactly one nominal sits and where this nominal can be any nominal of N'. Note that it is possible to construct this formula so that it is in $HGL_{r+1}(c,d)$ and contains no applications of $\Diamond^+$ or $\Box^+$. From above, it is not difficult to see that $\mathcal{A}_{m-1,r} \models \neg\Phi$ whereas $\mathcal{A}_{m,r} \not\models \neg\Phi$.

The following result is immediate from Theorem 4.5.6 and Lemmas 4.5.5, 4.5.7 and 4.5.8.

Theorem 4.5.9. *When $r \geq 0$, $c \geq 0$ and $d \geq 0$, it is the case that* HGL$_r(c,d) \subset$ HGL$_{r+1}(c,d)$.

4.5.2 Variable numbers of propositional symbols and nominals

Now the case is considered where letting the number of propositional symbols or nominals vary whilst keeping the quantifier-rank fixed is examined. For $m \geq 1$, define the digraph

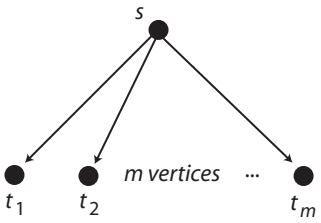

Figure 4.5: The digraph $\mathcal{H}_m$.

$\mathcal{H}_m$ as follows:

- the vertices of $\mathcal{H}_m$ are $\{s, t_1, t_2, \ldots, t_m\}$
- the edges of $\mathcal{H}_m$ are those edges of $\{(s, t_j) : 1 \leq j \leq m\}$.

Hence, $\mathcal{H}_m$ is a star with central vertex s and where all edges are directed away from s to m outer vertices, and can be visualized in Figure 4.5.

Theorem 4.5.10. *Let $r \geq 1$, $c \geq 0$ and $d \geq 0$. Define $m = d + 2^c$. The class of graphs containing just $\mathcal{H}_m$ is $\mathrm{HGL}_r(c,d)$-inexpressible, but that if $c' \geq c$, $d' \geq d$ and $c' + d' = c + d + 1$ then $\mathcal{H}_m$ is $\mathrm{HGL}_r(c',d')$-expressible. Thus, $\mathrm{HGL}_r(c,d) \subset \mathrm{HGL}_r(c',d')$.*

Proof. As before, vertices of $\mathcal{H}_m$ or $\mathcal{H}_{m+1}$ are denoted by using superscripts, as in $s^{\mathcal{H}_{m+1}}$ or $t_1^{\mathcal{H}_m}$, for example.

Let P be a set of c proposition symbols and let N be a set of d nominals. Consider a play of the r-round colouring $\mathrm{HGL}(\mathsf{P},\mathsf{N})$-game on the class of graphs containing $\mathcal{H}_m$. The Duplicator starts the game by choosing the structures $\mathcal{H}_m$ and $\mathcal{H}_{m+1}$. Spoiler makes a colouring-start move so as to build the pointed Kripke $\mathsf{P} \cup \mathsf{N}$-structure $(\mathcal{H}_{m+1}, \lambda, v)$. The *colour* of a vertex x of $(\mathcal{H}_{m+1}, \lambda)$ is the set of propositional symbols p of P for which $x \in \lambda(p)$ in union with the set of nominals n of N for which $x = \lambda(n)$ (and do likewise in other structures). In particular, there are two outer vertices of $(\mathcal{H}_{m+1}, \lambda)$, say $t_i^{\mathcal{H}_{m+1}}$ and $t_j^{\mathcal{H}_{m+1}}$, with identical colours. Duplicator replies and builds the pointed Kripke $\mathsf{P} \cup \mathsf{N}$-structure $(\mathcal{H}_m, \mu, u)$ by taking a copy of $(\mathcal{H}_{m+1}, \lambda, v)$ and deleting the vertex $t_i^{\mathcal{H}_m}$ (and its incident edge), as well as renaming v as u. Duplicator trivially has a winning strategy in the subsequent r-round $\mathrm{HGL}(\mathsf{P},\mathsf{N})$-game on $(\mathcal{H}_m, \mu, u)$ and $(\mathcal{H}_{m+1}, \lambda, v)$, and so by Theorem 4.4.7 $\mathcal{H}_m$ is $\mathrm{HGL}_r(c,d)$-inexpressible.

Alternatively, if Spoiler makes a colouring-start move so as to build the pointed $\mathsf{P} \cup \mathsf{N}$-structure $(\mathcal{H}_m, \mu, u)$ then Duplicator replies and builds the pointed $\mathsf{P} \cup \mathsf{N}$-structure $(\mathcal{H}_{m+1}, \lambda, v)$ by taking a copy of $(\mathcal{H}_m, \mu, u)$ and extending it with a new vertex $t^{\mathcal{H}_{m+1}}$ and edge $(s^{\mathcal{H}_{m+1}}, t^{\mathcal{H}_{m+1}})$ so that the colour of $t^{\mathcal{H}_{m+1}}$ is identical to the colour of some point $t_i^{\mathcal{H}_m}$ of $\mathcal{H}_m$ upon which no nominal sits and that is different from u. The point v is chosen as the point (corresponding to) u. Duplicator trivially has a winning strategy in the subsequent r-round $\mathrm{HGL}(\mathsf{P},\mathsf{N})$-game on $(\mathcal{H}_m, \mu, u)$ and $(\mathcal{H}_{m+1}, \lambda, v)$. Hence, $\mathcal{H}_m \equiv_{\mathrm{HGL}_r(c,d)} \mathcal{H}_{m+1}$ by Theorem 4.4.4.

Let P' be a set of c' propositional symbols and let N' be a set of d' nominals (with c' and d' as in the statement of the result). Consider a play of the r-round colouring

$\mathrm{HGL}(\mathsf{P}', \mathsf{N}')$-game in which Duplicator starts the game by choosing the structures $\mathcal{H}_m$ and $\mathcal{H}_{m+1}$. Suppose that Spoiler makes a colouring-start move so as to build the pointed Kripke $\mathsf{P} \cup \mathsf{N}$-structure $(\mathcal{H}_{m+1}, \lambda, v)$ where every outer vertex has a different colour and v is the vertex $s^{\mathcal{H}_{m+1}}$ (this is possible irrespective of whether $c' > c$ or $d' > d$). No matter which colouring-start move Duplicator replies with, so as to build $(\mathcal{H}_m, \mu, u)$, Duplicator must choose u as $s^{\mathcal{H}_m}$ (in order to stand a chance of winning the play) and there will exist some outer vertex of $(\mathcal{H}_{m+1}, \lambda)$ whose colour is not represented in $(\mathcal{H}_m, \mu)$. Spoiler clearly wins the subsequent r-round $\mathrm{HGL}(\mathsf{P}', \mathsf{N}')$-game on $(\mathcal{H}_m, \mu, u)$ and $(\mathcal{H}_{m+1}, \lambda, v)$ by making the appropriate $\Box$-move. This winning strategy characterises the structure $\mathcal{H}_m$ and so no matter which other structure Duplicator chooses, Spoiler can apply this strategy to win the game, hence by Theorem 4.4.7 $\mathcal{H}_m$ is $\mathrm{HGL}_r(c', d')$-expressible.

The result that $\mathrm{HGL}_r(c, d) \subset \mathrm{HGL}_r(c', d')$ immediately follows from $\mathcal{H}_m$ being both $\mathrm{HGL}_r(c, d)$-inexpressible and $\mathrm{HGL}_r(c', d')$-expressible. $\Box$

Finally, the situation for the logics $\mathrm{HGL}_0(c, d)$, where $c \geq 0$ and $d \geq 0$, is considered. Formulae of these logics do not involve the operators $\Diamond$, $\Diamond^+$, $\Box$, $\Box^+$ and $@_n$; that is, they are simply Boolean combinations of proposition symbols, nominals, $\top$ and $\bot$. Let φ be a formula of one of these logics. It is not difficult to see that:

- if $\mathcal{G}$ is a digraph with at least 2 vertices then $\mathcal{G} \models \varphi$ if, and only if, when regarding all propositional symbols and nominals in φ as Boolean variables, φ is a tautology
- if $\mathcal{G}$ is a digraph with 1 vertex then $\mathcal{G} \models \varphi$ if, and only if, when regarding all propositional symbols and nominals in φ as Boolean variables and make all Boolean variables corresponding to nominals true, the resulting formula φ is a tautology.

In particular, if φ is a formula of one of the above logics:

- if there exists a digraph $\mathcal{G}$ with at least 2 vertices such that $\mathcal{G} \models \varphi$ then the problem defined by φ consists of all digraphs
- if φ is not valid in any digraph with at least 2 vertices and no nominals appear in φ then the problem defined by φ is the empty problem
- if φ is not valid in any digraph with at least 2 vertices and at least 1 nominal appears in φ then either φ is valid in both digraphs with 1 vertex or neither (and both situations are possible).

Thus, the following result is obtained:

Theorem 4.5.11. *Let $c \geq 0$ and $d \geq 0$.*

- *For each $c \geq 0$, $\mathrm{HGL}_0(0, 0) = \mathrm{HGL}_0(c, 0)$, with the class of problems so defined consisting of the two problems consisting of all digraphs and of no digraphs.*
- *If $d \geq 1$, $\mathrm{HGL}_0(c, d)$ consists of three problems, namely the problems consisting of all digraphs, of no digraphs and of both the digraphs with 1 vertex.*

Theorems 4.5.2, 4.5.6, 4.5.9 and 4.5.10 give a full characterisation of the relative expressiveness of fragments of Hybrid Graph Logic that are obtained by varying the quantifier depth, number of proposition symbols and number of nominals.

Chapter 5

Conclusions

The research presented in this thesis examines the field of Finite Model Theory and its applications to Theoretical Computer Science. It has followed two distinct themes: the Descriptive Complexity of Optimisation Problems (Chapter 3) and the Expressiveness of Fragments of Hybrid Graph Logic (Chapter 4).

5.1 Descriptive Complexity of Optimisation Problems

The theory of Descriptive Complexity provides a strong link between Finite Model Theory and Computation Complexity. Most of the results obtained in the field have been with regards to characterisations of classes of decision problems, with few results extending to optimisation problems and those that do mainly focussing on NP-optimisation problems. Our focus was on polynomial-time optimisation problems and the development of frameworks for characterising them. The work started by analysing existing frameworks, which led to the results in Theorem 3.3.4 and Theorem 3.3.6, which state that a certain type of framework, the second-order tuple-counting framework, cannot characterise P-optimisation problems unless $\mathsf{P} = \mathsf{NP}$. Once this result was realised, we set about building frameworks from other candidate logics, most notably first-order logics augmented with fixed-point operators. This led to the full characterisation of P_{opt}^{PB} using a framework based around the inflationary fixed-point logic FO(IFP) in Theorem 3.4.1 and the least fixed-point logic FO(LFP) in Corollary 3.4.2. This framework is novel in that it uses the *inductive depth* of the fixed-point operator, rather than counting the number of tuples that satisfy a formula or measuring the cardinality of a relation, as had been done by previous frameworks.

These two results, the hardness of the tuple-counting framework, even when restricted to Horn formulae and the fixed-point framework characterising P_{opt}^{PB}, where presented by myself at the 5$^{\text{th}}$ International Computer Science Symposium in Kazan, Russia (CSR 2010) [AM10] and published in the conference proceedings [GS10]. Our paper was selected to appear in a special edition of Theory of Computer Systems (ToCS) and we have submitted a full length version for review and publication.

In Theorem 3.6.6 another framework is presented that characterises P_{opt}^{PB} using the measure of the cardinality of a relation. This result was surprising given the hardness of the very similar tuple-counting frameworks in Theorems 3.3.4 and 3.3.6 and has led to the conclusion that the tuple-counting part of the framework breaks the Horn condition

required in order to ensure the solution can be computed in polynomial-time. Future avenues of research could look at this complexity gap and investigate if any subsets of the tuple-counting framework, e.g. with limited tuple arity; unary quantifiers etc. resulted in a polynomial time computable solution. This would also tie in with the research that examines the tractability frontier from the perspective of logic [GKS04].

Section 3.7 examines the methods used to remove the polynomially-bounded requirement from the optimisation problems characterised by logical frameworks. Extending these results to P_{opt} for one of our frameworks would be an interesting line of future research.

5.2 Expressiveness of Fragments of Hybrid Graph Logic

The application of tools from Finite Model Theory to hybrid modal logic has resulted in a game that characterises the expressibility of properties in Hybrid Graph Logic. The game presented in Theorem 4.4.1 extends the Ehrenfeucht-Fraïssé style game for bisimulation to deal with both nominals and universal access. We then extended the game further to work for frame validity by using the idea of *colouring rounds* from the second-order Ehrenfeucht-Fraïssé game first developed by Fagin [Fag75]. The aim of the research was to search for classes of structures and develop winning strategies on them for the Duplicator in order to show that these structures are expressible in one fragment of Hybrid Graph Logic but inexpressible in another.

We used this game, structures and winning strategies to completely characterise the relative expressiveness of fragments of Hybrid Graph Logic that are obtained by restriction the quantifier-rank, the number of proposition symbols and the number of nominals. The hierarchy results are presented in Theorems 4.5.2, 4.5.6, 4.5.9 and 4.5.10.

Directions for future research in this area could include extending these hierarchy results to more expressive hybrid logics, such as those involving the binder $\downarrow$ and also to undirected graphs. It would also be interesting to examine the relative expressiveness of fragments that are parameterised by the number of alternations of modal quantifiers, rather than just the quantifier depth, i.e. is the fragment that contains a block of $\Diamond$ quantifiers followed by a block of $\Box$ quantifiers strictly more expressive than a fragment that just contains (either) $\Diamond$ or $\Box$ quantifiers?

The development of winning strategies for the Duplicator in these games is of key importance, and it may be the case that the techniques used in devising winning strategies presented in this thesis could be applied elsewhere.

Bibliography

[ABM00] Carlos Areces, Patrick Blackburn, and Maarten Marx, *The computational complexity of hybrid temporal logics*, Logic Journal of the IGPL **8** (2000), no. 5, 653–679.

[ABM01] ———, *Hybrid logics: characterization, interpolation and complexity*, J. Symb. Log. **66** (2001), no. 3, 977–1010.

[AF90] Miklós Ajtai and Ronald Fagin, *Reachability is harder for directed than for undirected finite graphs*, J. Symb. Log. **55** (1990), no. 1, 113–150.

[AF97] Sanjeev Arora and Ronald Fagin, *On winning strategies in Ehrenfeucht-Fraïssé games*, Theor. Comput. Sci. **174** (1997), no. 1-2, 97–121.

[AM10] Farid Ablayev and Ernst W. Mayr (eds.), *5th international computer science symposium in Russia, CSR 2010, Kazan, Russia, June 16-20, 2010, proceedings*, Lecture Notes in Computer Science, vol. 6072, Springer Berlin, 2010.

[AtC07] Carlos Areces and Balder ten Cate, *Hybrid logics*, Handbook of Modal Logic (Patrick Blackburn, Johan Van Benthem, and Frank Wolter, eds.), Studies in Logic and Practical Reasoning, vol. 3, Elsevier, 2007, pp. 821 – 868.

[AW00] Joan M. Aldous and Robin J. Wilson, *Graphs and applications: an introductory approach*, Springer, 2000.

[BBJ02] George S. Boolos, John P. Burgess, and Richard C. Jeffrey, *Computability and logic*, fourth ed., Cambridge University Press, 2002.

[BCG01] Elwyn R. Berlekamp, John H. Conway, and Richard K. Guy, *Winning ways for your mathematical plays*, second ed., vol. 1, A K Peters, Natick, Massachusetts, 2001.

[BdRV01] Patrick Blackburn, Maarten de Rijke, and Yde Venema, *Modal logic*, Cambridge tracts in theoretical computer science, vol. 53, Cambridge University Press, 2001.

[BL99] Hans Kleine Büning and Theodor Lettman, *Propositional logic: deduction and algorithms*, Cambridge tracts in theoretical computer science, vol. 48, Cambridge University Press, 1999.

[BM08] Orestes Bueno and Prabhu Manyem, *Polynomial-time maximisation classes: syntactic hierarchy*, Fundamenta Informaticae **84** (2008), no. 1, 111–133.

[Bol98] Béla Bollobás, *Modern graph theory*, Graduate texts in mathematics, vol. 184, Springer, 1998.

[BS09] Mario R. F. Benevides and L. Menasché Schechter, *Using modal logics to express and check global graph properties*, Logic Journal of the IGPL **17** (2009), no. 5, 559–587.

[CJ91] Christian Choffrut and Matthias Jantzen (eds.), *STACS 91, 8th annual symposium on theoretical aspects of computer science, Hamburg, Germany, February 14-16, 1991, proceedings*, Lecture Notes in Computer Science, vol. 480, Springer, 1991.

[CLRS09] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein, *Introduction to algorithms*, third ed., MIT Press, 2009.

[Coo71] Stephen A. Cook, *The complexity of theorem-proving procedures*, in Harrison et al. [HBU71], pp. 151–158.

[CS01] Edmund M. Clarke and Bernd-Holger Schlingloff, *Chapter 24 – model checking*, Handbook of Automated Reasoning (Alan Robinson and Andrei Voronkov, eds.), North-Holland, Amsterdam, 2001, pp. 1635 – 1790.

[DBL82] *Proceedings of the fourteenth annual ACM symposium on theory of computing, 5-7 May 1982, San Francisco, California, USA*, ACM, 1982.

[Die97] Reinhard Diestel, *Graph theory*, Graduate texts in mathematics, vol. 173, Springer, 1997.

[dR87] M. de Rougemont, *Second-order and inductive definability on finite structures*, Zeitschrift für Mathematische Logik und Grundlagen der Mathematik **33** (1987), 47–63.

[EF99] Heinz-Dieter Ebbinghaus and Jörg Flum, *Finite model theory*, second ed., Monographs in mathematics, Springer, 1999.

[EH85] E. Allen Emerson and Joseph Y. Halpern, *Decision procedures and expressiveness in the temporal logic of branching time*, J. Comput. Syst. Sci. **30** (1985), no. 1, 1–24.

[Ehr61] A. Ehrenfeucht, *An application of games to the completeness problem for formalized theories*, Fund. Math. **49** (1961), 129–141.

[Fag74] Ronald Fagin, *Generalized first-order spectra and polynomial-time recognizable sets*, Complexity and Computation, SIAM-AMS Proceedings, vol. 7, 1974, pp. 43–73.

[Fag75] ———, *Monadic generalized spectra*, Zeitschrift für Mathematische Logik und Grundlagen der Mathematik **21** (1975), 89–96.

[Fag96] ———, *Easier ways to win logical games*, in Immerman and Kolaitis [IK96], pp. 1–32.

[FdR06] Massimo Franceschet and Maarten de Rijke, *Model checking hybrid logics (with an application to semistructured data)*, J. Applied Logic **4** (2006), no. 3, 279–304.

[FG06] Jörg Flum and Martin Grohe, *Parameterized complexity theory*, Texts in theoretical computer science; an EATCS series, Springer, 2006.

[Fra54] R. Fraïssé, *Sur quelques classifications des systèmes de relations*, Publ. Sci. Univ. Alger. Sér. A **1** (1954), 35–182.

[FSV95] Ronald Fagin, Larry J. Stockmeyer, and Moshe Y. Vardi, *On monadic NP vs. monadic co-NP*, Inf. Comput. **120** (1995), no. 1, 78–92.

[GH10] Jürgen Giesl and Reiner Hähnle (eds.), *Automated reasoning, 5th international joint conference, IJCAR 2010, Edinburgh, UK, July 16-19, 2010, proceedings*, Lecture Notes in Computer Science, vol. 6173, Springer, 2010.

[GHR95] Raymond Greenlaw, H. James Hoover, and Walter L. Ruzzo, *Limits to parallel computation: P-completeness theory*, Oxford University Press, 1995.

[GJ79] Michael R. Garey and David S. Johnson, *Computers and intractability, a guide to the theory of NP-completeness*, W. H. Freeman and Company, 1979.

[GJS76] M. R. Garey, David S. Johnson, and Larry J. Stockmeyer, *Some simplified NP-complete graph problems*, Theor. Comput. Sci. **1** (1976), no. 3, 237–267.

[GKL+07] Erich Grädel, Phokion G. Kolaitis, Leonid Libkin, Maarten Marx, Yde Venema, Joel Spencer, Moshe Y. Vardi, and Scott Weinstein, *Finite model theory and its applications*, Texts in theoretical computer science; an EATCS series, Springer, 2007.

[GKS04] Georg Gottlob, Phokion G. Kolaitis, and Thomas Schwentick, *Existential second-order logic over graphs: charting the tractability frontier*, J. ACM **51** (2004), no. 2, 312–362.

[GO07] Valentin Goranko and Martin Otto, *Model theory of modal logic*, Handbook of Modal Logic (Patrick Blackburn, Johan Van Benthem, and Frank Wolter, eds.), Studies in Logic and Practical Reasoning, vol. 3, Elsevier, 2007, pp. 249 – 329.

[Grä91a] Erich Grädel, *Capturing complexity classes by fragments of second order logic*, Structure in Complexity Theory Conference, 1991, pp. 341–352.

[Grä91b] ———, *The expressive power of second order horn logic*, in Choffrut and Jantzen [CJ91], pp. 466–477.

[Grä92] ———, *Capturing complexity classes by fragments of second-order logic*, Theor. Comput. Sci. **101** (1992), no. 1, 35–57.

[Grä07] ———, *Finite model theory and descriptive complexity*, ch. 3, pp. 125–230, in Brauer et al. [GKL+07], 2007.

[GS86] Yuri Gurevich and Saharon Shelah, *Fixed-point extensions of first-order logic*, Annals of Pure and Applied Logic **32** (1986), 265–280.

[GS10] James Gate and Iain A. Stewart, *Frameworks for logically classifying polynomial-time optimisation problems*, in Ablayev and Mayr [AM10], pp. 120–131.

[HA50] D. Hilbert and W. Ackermann, *Principles of mathematical logic*, Chelsea Publishing Company, New York, 1950.

[HBU71] Michael A. Harrison, Ranan B. Banerji, and Jeffrey D. Ullman (eds.), *Proceedings of the 3rd annual ACM symposium on theory of computing, May 3-5, 1971, Shaker Heights, Ohio, USA*, ACM, 1971.

[HC96] G. E. Hughes and M. J. Cresswell, *A new introduction to modal logic*, Routledge, 1996.

[IK96] Neil Immerman and Phokion G. Kolaitis (eds.), *Descriptive complexity and finite models, proceedings of a DIMACS workshop, January 14-17, 1996, Princeton University*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, vol. 31, American Mathematical Society, 1996.

[Imm82] Neil Immerman, *Relational queries computable in polynomial time (extended abstract)*, in *STOC* [DBL82], pp. 147–152.

[Imm86] ———, *Relational queries computable in polynomial time*, Information and Control **68** (1986), no. 1-3, 86–104.

[Imm88] ———, *Nondeterministic space is closed under complementation*, SIAM J. Comput. **17** (1988), no. 5, 935–938.

[Imm99] ———, *Descriptive complexity*, Graduate texts in computer science, Springer, 1999.

[JLL76] Neil D. Jones, Y. Edmund Lien, and William T. Laaser, *New problems complete for nondeterministic log space*, Mathematical Systems Theory **10** (1976), 1–17.

[JS87] Brigitte Jaumard and Bruno Simeone, *On the complexity of the maximum satisfiability problem for horn formulas*, Inf. Process. Lett. **26** (1987), no. 1, 1–4.

[KKM94] Rajeev Kohli, Ramesh Krishnamurti, and Prakash Mirchandani, *The minimum satisfiability problem*, SIAM J. Discrete Math. **7** (1994), no. 2, 275–283.

[Knu73] Donald E. Knuth, *The art of computer programming, volume I: fundamental algorithms, 2nd edition*, Addison-Wesley, 1973.

[Kol07] Phokion G. Kolaitis, *On the expressive powers of logics on finite models*, ch. 2, pp. 27–123, in Brauer et al. [GKL$^+$07], 2007.

[Kre04] Stephan Kreutzer, *Expressive equivalence of least and inflationary fixed-point logic*, Ann. Pure Appl. Logic **130** (2004), no. 1-3, 61–78.

[KS10] Mark Kaminski and Gert Smolka, *Terminating tableaux for hybrid logic with eventualities*, in Giesl and Hähnle [GH10], pp. 240–254.

[KT94] Phokion G. Kolaitis and Madhukar N. Thakur, *Logical definability of NP optimization problems*, Information and Computation **115** (1994), no. 2, 321–353.

[KT95] ———, *Approximation properties of NP minimization classes*, Journal of Computer and System Sciences **50** (1995), no. 3, 391–411.

[KV07] Phokion G. Kolaitis and Moshe Y. Vardi, *A logical approach to constraint satisfaction*, ch. 6, pp. 339–370, in Brauer et al. [GKL$^+$07], 2007.

[Lib04] Leonid Libkin, *Elements of finite model theory*, Texts in theoretical computer science; an EATCS series, Springer, Berlin, 2004.

[Man08] Prabhu Manyem, *Syntactic characterizations of polynomial time optimization classes*, Chicago Journal of Theoretical Computer Science **2008** (2008), no. 3, 1–23.

[MMS$^+$10] Arne Meier, Martin Mundhenk, Thomas Schneider, Michael Thomas, Volker Weber, and Felix Weiss, *The complexity of satisfiability for fragments of hybrid logic – part I*, J. Applied Logic **8** (2010), no. 4, 409–421.

[MV07] Maarten Marx and Yde Venema, *Local variations on a loose theme: modal logic and decidability*, ch. 7, pp. 371–429, in Brauer et al. [GKL$^+$07], 2007.

[Nur96] Juha Nurmonen, *On winning strategies with unary quantifiers*, J. Log. Comput. **6** (1996), no. 6, 779–798.

[Pap94] Christos M. Papadimitriou, *Computational complexity*, Addison-Wesley, 1994.

[Pla84] David A. Plaisted, *Complete problems in the first-order predicate calculus*, J. Comput. Syst. Sci. **29** (1984), no. 1, 8–35.

[PR93] Alessandro Panconesi and Desh Ranjan, *Quantifiers and approximation*, Theoretical Computer Science **107** (1993), no. 1, 145–163.

[PY91] Christos H. Papadimitriou and Mihalis Yannakakis, *Optimization, approximation, and complexity classes*, Journal of Computer and System Science **43** (1991), 425–440.

[RRR98] Venkatesh Raman, Bala Ravikumar, and S. Srinivasa Rao, *A simplified NP-complete MAXSAT problem*, Inf. Process. Lett. **65** (1998), no. 1, 1–6.

[Sav70] Walter J. Savitch, *Relationships between nondeterministic and deterministic tape complexities*, J. Comput. Syst. Sci. **4** (1970), no. 2, 177–192.

[Sch96] Thomas Schwentick, *On winning ehrenfeucht games and monadic NP*, Ann. Pure Appl. Logic **79** (1996), no. 1, 61–92.

[Sip06] Michael Sipser, *Introduction to the theory of computation*, second ed., Thomson Course Technology, 2006.

[SST95] Sanjeev Saluja, K. V. Subrahmanyam, and Madhujar N. Thakur, *Descriptive complexity of #P functions*, Journal of Computer and System Sciences **50** (1995), 493–505.

[Sze87] Róbert Szelepcsényi, *The method of forcing for nondeterministic automata*, Bulletin of the EATCS **33** (1987), 96–99.

[Top12] *TOP500 supercomputing sites*, http://top500.org, September 2012.

[Tov84] Craig A. Tovey, *A simplified NP-complete satisfiability problem*, Discrete Applied Mathematics **8** (1984), no. 1, 85–89.

[Var82] Moshe Y. Vardi, *The complexity of relational query languages (extended abstract)*, in *STOC* [DBL82], pp. 137–146.

[Var96] ———, *Why is modal logic so robustly decidable?*, in Immerman and Kolaitis [IK96], pp. 149–184.

[Vol99] Heribert Vollmer, *Introduction to circuit complexity*, Texts in theoretical computer science; an EATCS series, Springer, 1999.

[Zim98] Marius Zimand, *Weighted NP optimization problems: logical definability and approximation properties*, SIAM Journal on Computing **28** (1998), no. 1, 36–56.

[Zoo12] *Complexity zoo*, http://qwiki.stanford.edu/index.php/Complexity_Zoo, September 2012.